SPARCserver 490 (SunOS 4.0.3) Release Manual

Sun Microsystems, Inc. • 2550 Garcia Avenue • Mountain View, CA 94043 • 415-960-1300

Part No: 800-3473-10
Revision A of 11 December 1989

The Sun logo, Sun Microsystems, Sun Workstation, NFS, and TOPS are registered trademarks of Sun Microsystems, Inc.

Sun, Sun-2, Sun-3, Sun-4, Sun386*i*, SPARCstation, SPARCserver, NeWS, NSE, OpenWindows, SPARC, SunInstall, SunLink, SunNet, SunOS, SunPro, and SunView are trademarks of Sun Microsystems, Inc.

UNIX is a registered trademark of AT&T; OPEN LOOK is a trademark of AT&T.

All other products or services mentioned in this document are identified by the trademarks or service marks of their respective companies or organizations, and Sun Microsystems, Inc. disclaims any responsibility for specifying which marks are owned by which companies or organizations.

LEGAL NOTICE TO USERS:

Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc., and may also be a trademark of various telephone companies around the world. Sun will be revising future versions of software and documentation to remove references to Yellow Pages.

Copyright © 1989 Sun Microsystems, Inc. – Printed in U.S.A.

All rights reserved. No part of this work covered by copyright hereon may be reproduced in any form or by any means – graphic, electronic, or mechanical – including photocopying, recording, taping, or storage in an information retrieval system, without the prior written permission of the copyright owner.

Restricted rights legend: use, duplication, or disclosure by the U.S. government is subject to restrictions set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013 and in similar clauses in the FAR and NASA FAR Supplement.

The Sun Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees.

This software and documentation is based in part on the Fourth Berkeley Software Distribution under license from the Regents of the University of California. We acknowledge the following individuals and institutions for their role in its development: The Regents of the University of California, the Electrical Engineering and Computer Sciences Department at the Berkeley Campus of the University of California, and Other Contributors.

Contents

Chapter 1 Introduction 3

1.1. About This Manual 3

1.2. Media Box Contents 4

1.3. Other Documentation 5

Release Documentation Box Contents 6

Beginner's Guides 6

Reference Manuals 6

System Administration Manuals 6

Programmer's Guides 7

SPARC Documents 7

1.4. Typographic Conventions 7

1.5. Machine Architecture Terminology 8

Chapter 2 Installing SPARCserver 490 Software 11

2.1. Installation Overview 11

2.2. Local Installation 12

If You Use yp 12

Saving Defect Lists 15

Do not copy miniroot to partition "a" 16

If You Have a Second Ethernet 18

Changing root and swap Partitions 19

The /export Partitions 19

Freehog Partition for Heterogeneous Installations 19

About the Preserve Option 19

If You Don't Assign All Disks 20
Tape Device Error Message 21
If You Will Build a Customized Kernel 21
About the usr Partition 21
If You Use SunDials 21
Other Architectures — Sun-4c Last 22
2.3. Post-Installation Procedures 26
You Must Edit /etc/netmasks 26
Install Small Kernels 26
Halt from the Miniroot 26
Set EEPROM for IPI Booting 26
Boot System in Single-User Mode 26
Customize Your Kernel 27
Configure an Ethernet Gateway 27
You Must Edit /etc/netmasks if You Have a Second Ethernet 27
If Your SPARCserver 490 Is a yp Server 27
Setting yp Maps for yp Server 28
If Your SPARCserver 490 Is a yp Client 28
Boot System Multi-User 28
2.4. Dataless Client Installation 28
2.5. Adding Support for Other Architectures After Installation 29
If You Add Sun-4c and Other Architectures Together 29
Reinstall Sun-4c if You Add Other Architectures Later 30
Reverting to a Previous Version of vmunix 31
2.6. Remote Installation 31
Requirements for performing a remote installation 31
Warning: these activities can be dangerous; the procedure must be followed very carefully. 33
2.7. Tape Contents 38
SunOS 4.0.3 Installation Tape Table of Contents 38
SPARCserver 490 Installation Tape Table of Contents 39
Extracting the Table of Contents from a Tape 40
Listing Files in a Software Category 41

Chapter 3 Kernel Configuration **45**
3.1. Supplied Kernel Configuration Files 45
The GENERIC Kernel 45
The SPARCserver 490 Kernel: IDST490 45
3.2. Customizing a Kernel 50
Create /dev Entries 50
Make Entries in /etc/ttytab 51
Possible Error Messages 52
Monitoring Performance 52
3.3. Configuring a Kernel 53

Chapter 4 IPI8-1000 Disk Drive and ISP-80 Controller **57**
4.1. Description and Configuration Options 57
Definition of Terms 57
4.2. Addressing and Naming Conventions 57
Physical Addresses 57
Device Names 58
Boot Address Format 58
4.3. Default Boot from IPI8-1000 Disk 58
Using the q Command 58
Using the eeprom Command 59
4.4. Rebooting the System After Swapping a Disk Drive 61
4.5. IPI Error Messages 61
Basic Error Message Forms 61
Errors where the drive can be identified and there is a particular disk partition in use 61
For operations on the whole drive (such as format), where no partition information would be relevant 61
For operations to the controller, where the drive number isn't significant 62
<operation> field 62
Messages from the IPI Controller or IPI Drive 62
Message/Microcode Exception (from IPI Controller) 62
Intervention Required (from IPI Drive) 63

Alternate Port Exception (from IPI Drive) 63
Machine Exception (from IPI Drive/Controller) 63
Command Exception (from IPI Controller) 63
Conditional Success (from IPI Drive/Controller) 63
Incomplete (from IPI Controller) 63
Errors reported by the IPI Driver 63

Chapter 5 SCSI Devices **69**
5.1. SCSI Tape Drive Configuration Options 69
5.2. Front-Load 1/2" Tape Drive 69
5.3. 2.3-Gbyte 8mm Tape Drive 69
5.4. 150-Mbyte 1/4" Tape Drive 69
5.5. Determining the Tape Drive Device Name 69
Tape Drive Configurations 70
Using the mt Command 70
5.6. Data Format Addressing 71
150-Mbyte 1/4" Tape Drive 71
2.3-Gbyte 8mm Tape Drive 71
Front-Load 1/2" Tape Drive 71
5.7. Boot Address Formats 72

Chapter 6 SunNet Ethernet/VME Controller **75**
6.1. Description and Configuration 75
6.2. Configuring an Ethernet Gateway 75

Chapter 7 Diagnostic Testing **79**
7.1. Enhanced Diagnostic Software 79
The sundiag Program 79

Appendix A PROM User's Manual Addenda and Errata (2) **85**
A.1. New PROM Monitor Features 85
Entering Decimal Values 85
Entering ASCII Characters 85
New `^i` Feature 85

A.2. New Boot-Up Features 86
The Power-Up Display 86
Revising the Banner 86
Automatic IPI Boot-Up 88
Sun-3/80 Diagnostic Boot-Up 88
PROM Monitor Security Feature 89
The Password 89
Changing Security Modes 91
EEPROM Layout for PROM Security 91

Appendix B Man Pages for SPARCserver 490 (SunOS 4.0.3) Release **95**

Index **97**

1

Introduction

Introduction 3

1.1. About This Manual 3
1.2. Media Box Contents 4
1.3. Other Documentation 5
Release Documentation Box Contents 6
Beginner's Guides 6
Reference Manuals 6
System Administration Manuals 6
Programmer's Guides 7
SPARC Documents 7
1.4. Typographic Conventions 7
1.5. Machine Architecture Terminology 8

1

Introduction

This manual describes the SPARCserver 490 software release (SunOS 4.0.3), and explains how to install the software. Be sure to read the *Read This First* that accompanies this manual in the release box; because of publication deadlines, it may contain information that supersedes information in this manual. It also contains descriptions of known problems with the software, and tells you to get help if you need it.

The SPARCserver 490 is a high performance server based on the SPARC (Scalable Processor ARChitecture) chip technology. The SPARCserver 490 can act as a file, database, or compute server on a network or as a standalone processor serving a large number of time-sharing users. This release includes support for the following new hardware devices, described in the chapter listed after the device name:

- ISP-80 Controller and IPI8-1000 Disk Drive (Chapter 4)
- Front-Load 1/2-inch Tape Drive (Chapter 5)
- 2.3-Gbyte 8mm Tape Drive (Chapter 5)
- 150-Mbyte 1/4-inch Tape Drive (Chapter 5)
- SunNet Ethernet/VME Controller (Chapter 6)

The Release Manual also includes new or modified "man" pages and descriptions of changes to diagnostic tests.

See the appropriate hardware manuals for detailed descriptions of the hardware.

1.1. About This Manual

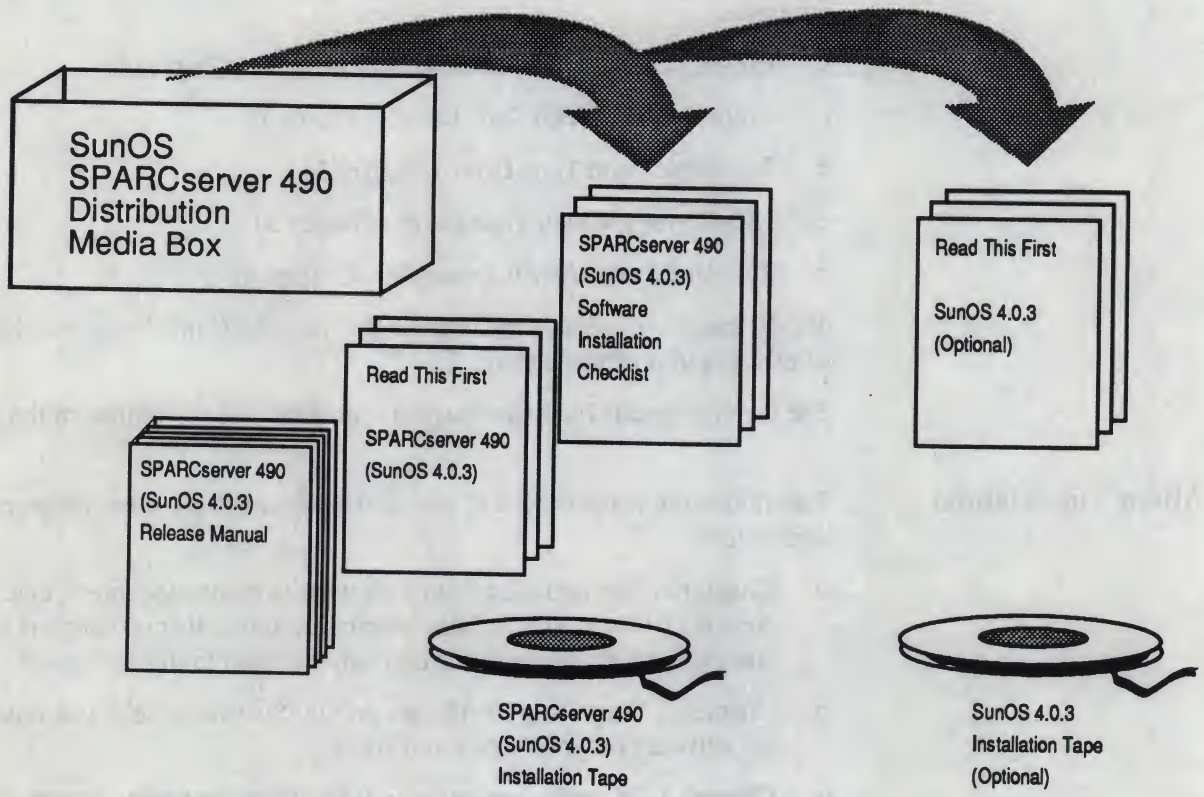
This document is comprised of this introduction and six other chapters, plus two appendices:

- Chapter 1, "Introduction," outlines what is in this document, describes the contents of the media and documentation boxes that accompany this release, and explains the typographic conventions used in this document.
- Chapter 2, "Installing SPARCserver 490 Software," tells you how to install the software on your server and clients.
- Chapter 3, "Kernel Configuration," describes the kernel configuration files supplied with this release, and explains how to customize your kernel for optimum performance.

- Chapter 4, "IPI8-1000 Disk Drive and ISP-80 Controller," describes the new high-speed, high-capacity disk drives and controllers available on your SPARCserver 490.
- Chapter 5, "SCSI Devices," describes the three tape drives available on your SPARCserver 490.
- Chapter 6, "SunNet Ethernet/VME Controller," describes the Ethernet controller which allows you to configure a second Ethernet port as a network gateway.
- Chapter 7, "Diagnostic Testing," describes Sundiag, a tool for testing your SPARCserver 490.
- Appendix A, "PROM User's Manual Addenda and Errata (2)," contains new and changed pages for insertion in the *PROM User's Manual.*
- Appendix B, "Man Pages," contains new and changed *SunOS Reference Manual* pages, for insertion in the Sun System Reference Library.

1.2. Media Box Contents

The contents of the media box are shown in the picture below. Note that the SunOS 4.0.3 installation tape and *Read This First SunOS 4.0.3* are optional; they will be included in the box only if you ordered your SPARCserver 490 with the SunOS 4.0.3 options.

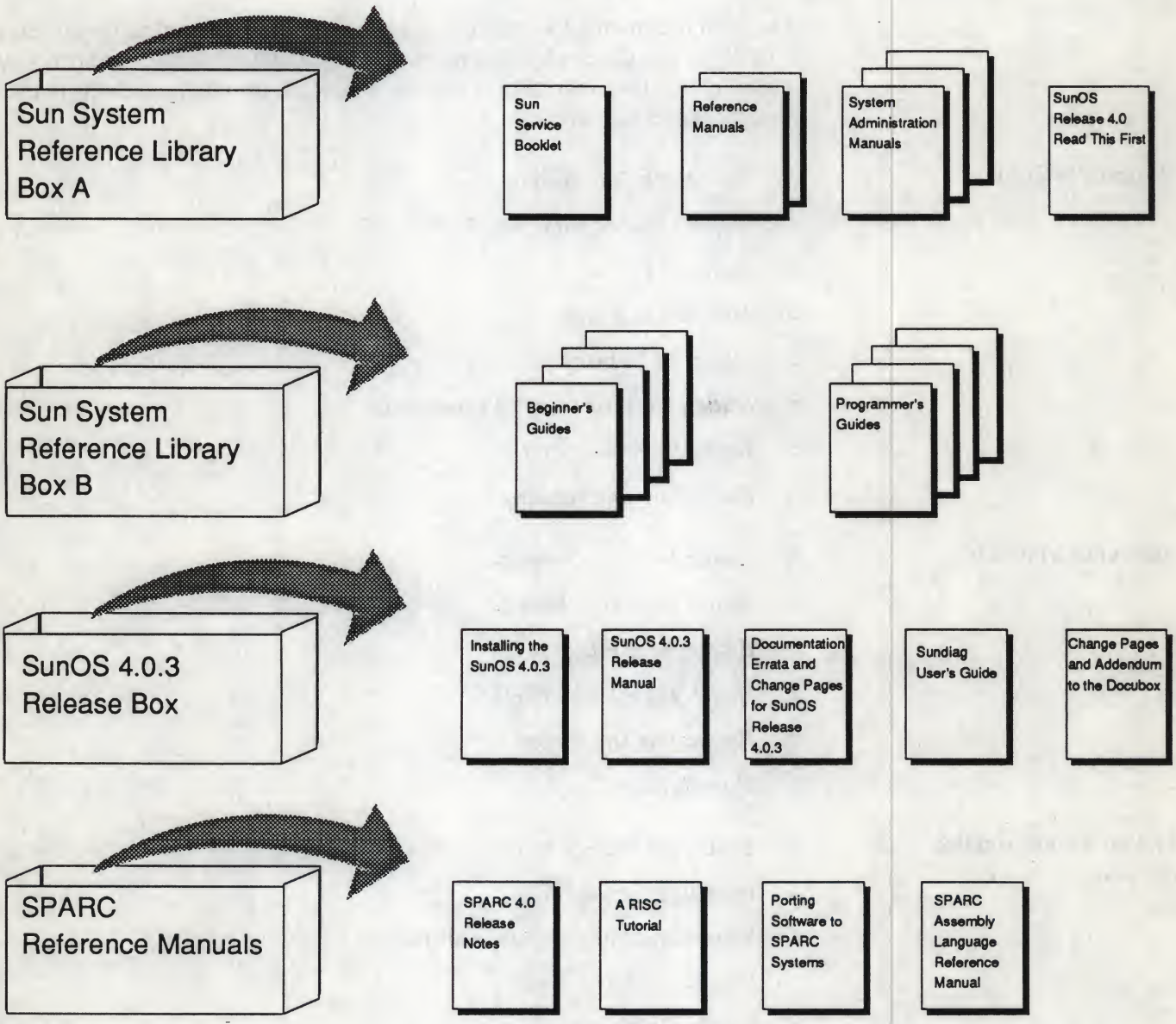


If you ordered SunOS 4.0.3 for other architectures, you will receive one box for each architecture you ordered (Sun-3, Sun-3x, Sun-4c). Each box will contain the tapes necessary to install SunOS 4.0.3 for a particular architecture.

If any of the tapes or documents was not in the box when it was delivered, call Sun at 1-800-USA-4SUN.

1.3. Other Documentation

If you ordered your SPARCserver 490 with the SunOS 4.0.3 documentation option, you will also receive Sun's System Reference Library (two boxes), the SunOS 4.0.3 Release Box, and the SPARC Reference Manuals.



Release Documentation Box Contents

During installation of SPARCserver 490 software, you may wish to refer to the documentation in the Release Documentation box — especially the *Installing the SunOS 4.0.3* manual, which supplements the installation instructions in Chapter 2 of this manual. The Release Documentation box contains:

- *Installing the SunOS 4.0.3* (PN 800-3812)
- *SunOS 4.0.3 Release Manual* (PN 800-3815)
- *Addenda & Errata to the Docubox* (PN 800-3378)
- *Sundiag User's Guide* (PN 800-3817)
- *Documentation Errata for SunOS Release 4.0.3* (PN 800-3377)

The other documents are broken down into five major categories; the contents of each of the categories are listed below. See the *Global Index* if you are not sure where to look for a particular document or subject; the *Global Index* is in the box marked "Reference Manuals."

Beginner's Guides

- *Sun System Introduction*
- *Getting Started with SunOS*
- *SunView 1*
- *Mail and Messages*
- *Using the Network*
- *Setting Up Your SunOS Environment*
- *Doing More with SunOS*
- *Self Help with Problems*

Reference Manuals

- *System Services Overview*
- *SunOS Reference Manual*
- *Editing Text Files*
- *Using NROFF and TROFF*
- *Formatting Documents*
- *Global Index*

System Administration Manuals

- *SunOS 4.0 Change Notes*
- *Installing the SunOS*
- *System and Network Administration*
- *Security Features Guide*
- *PROM User's Manual*
- *Sun System Diagnostics Manual*

Programmer's Guides

- *C Programmer's Guide*
- *Assembly Language Reference for Sun-2 and Sun-3*
- *Debugging Tools*
- *Floating Point Programmer's Guide*
- *Network Programming*
- *Programming Utilities and Libraries*
- *Writing Device Drivers*
- *SunView 1 Programmer's Guide*
- *SunView 1 System Programmer's Guide*
- *Pixrect Reference Manual*
- *Sun CGI Reference Manual*
- *SunCore Reference Manual*

SPARC Documents

- *SPARC 4.0 Release Notes*
- *A RISC Tutorial*
- *Porting Software to SPARC Systems*
- *SPARC-4 Assembly Language Reference Manual*

1.4. Typographic Conventions

The following typographic conventions are used in this manual.

- **bold font**

The **bold font** is used in step-by-step, numbered instructions, to indicate what you should do. An explanation of the step may follow in a normal font.

1. **Do this.**

   Read this explanation.

2. **Then do this.**

The **bold font** is also used in glossary-like explanations of terms to make the term being described stand out.

- `listing font`

This `font` is used to indicate system responses displayed on the screen, for literal values such as program, function, procedure, or variable names, and for listings of the contents of a file.

- boxed entries

Text within a box represents screen displays or the listing of the contents of a file. There is no user interaction shown in such displays.

```
line 1
line 2
```

- **bold listing font**

This **font** represents text that you type.

- Gray boxes

Material within a gray box represents a dialog you have with the system: in response to a system prompt or other activity, you type in something. The bold text in a gray box represents your response.

```
Enter name of client machine:  opusone
Enter name of server:  fourfox
```

- *italic font*

The *italic font* represents a variable for which you need to substitute the proper value; this font is also used for *emphasis* or to introduce a *new term* into the text.

```
% ypmatch hostname hosts
```

1.5. Machine Architecture Terminology

At various places in this and other documents, the term "system architecture," or "machine architecture," is used. Different architectures (Sun-2, Sun-3, Sun-3x, Sun-4, Sun-4c) refer to the type of CPU in the machine (68010, 68020, 68030, etc.). When performing a system installation or upgrade, you may need to know how to translate a system architecture into a CPU type or a machine name. The following chart correlates the three terms.

Architecture	CPU Type	Machine Name
Sun-2	68010	Sun-2/xx, Sun-2/1xx
Sun-3	68020	Sun-3/xx, Sun-3/1xx, Sun-3/2xx
Sun-3x	68030	Sun-3/80, Sun-3/4xx
Sun-4	SPARC	Sun-4/1xx, Sun-4/2xx, SPARCstation 3xx
Sun-4c	SPARC	SPARCstation 1

The SPARCserver 490 is a Sun-4 machine — a machine with a SPARC CPU board; it is sometimes called a Sun-4/490. Other Sun-4 machines have names such as Sun-4/110, Sun-4/260, and SPARCstation 330.

2

Installing SPARCserver 490 Software

Installing SPARCserver 490 Software **11**

2.1. Installation Overview 11
2.2. Local Installation 12
If You Use yp 12
Saving Defect Lists 15
Do not copy miniroot to partition "a" 16
If You Have a Second Ethernet 18
Changing root and swap Partitions 19
The /export Partitions 19
Freehog Partition for Heterogeneous Installations 19
About the Preserve Option 19
If You Don't Assign All Disks 20
Tape Device Error Message 21
If You Will Build a Customized Kernel 21
About the usr Partition 21
If You Use SunDials 21
Other Architectures — Sun-4c Last 22
2.3. Post-Installation Procedures 26
You Must Edit /etc/netmasks 26
Install Small Kernels 26
Halt from the Miniroot 26
Set EEPROM for IPI Booting 26
Boot System in Single-User Mode 26

Customize Your Kernel 27
Configure an Ethernet Gateway 27
You Must Edit /etc/netmasks if You Have a Second Ethernet 27
If Your SPARCserver 490 Is a yp Server 27
Setting yp Maps for yp Server 28
If Your SPARCserver 490 Is a yp Client 28
Boot System Multi-User 28
2.4. Dataless Client Installation 28
2.5. Adding Support for Other Architectures After Installation 29
If You Add Sun-4c and Other Architectures Together 29
Reinstall Sun-4c if You Add Other Architectures Later 30
Reverting to a Previous Version of vmunix 31
2.6. Remote Installation 31
Requirements for performing a remote installation 31
Warning: these activities can be dangerous; the procedure must be followed very carefully. 33
2.7. Tape Contents 38
SunOS 4.0.3 Installation Tape Table of Contents 38
SPARCserver 490 Installation Tape Table of Contents 39
Extracting the Table of Contents from a Tape 40
Listing Files in a Software Category 41

2

Installing SPARCserver 490 Software

Use the instructions here as an outline or guide to installing SunOS 4.0.3 software for the SPARCserver 490. See the appropriate hardware manuals for detailed information about the hardware.

For detailed instructions and explanations about preparing for an installation and about the installation process itself, see *Installing the SunOS 4.0.3*, which is in the documentation minibox included with this release; specific references to that document are included at various times in this chapter.

This chapter includes the following sections:

- Installation Overview
- Local Installation
- Post-Installation Procedures
- Dataless Client Installation
- Adding Support for Other Architectures After Installation
- Remote Installation
- Tape Contents

2.1. Installation Overview

To install the SPARCserver 490 (SunOS 4.0.3) software, you will do the following:

- Gather needed information (hostnames, Ethernet addresses, domain names, internet addresses, disk partition layouts, location of root, swap, and other directories) for the server and any clients to be installed.
- Format disks, modify disk partitioning, and label disks (if necessary).
- Copy miniroot from SPARCserver 490 tape to disk and boot the miniroot; start SunInstall.
- Complete SunInstall forms for SPARCserver 490 and all clients, if any.
- Install SunOS 4.0.3 on SPARCserver 490 and all clients, if any.

  Since the SPARCserver 490 is a Sun-4 machine, you must have SunOS 4.0.3 Sun-4 tapes available during the installation. If you are installing your SPARCserver 490 as a heterogeneous server (serving clients of more than

one architecture), you will also need to have the appropriate SunOS 4.0.3 tapes for those architectures available during the installation.

- Complete the installation by loading the SPARCserver 490 tape and running `sunupgrade` to install SPARCserver 490 files.
- Perform various post-installation steps such as installing small kernels for client machines, setting your EEPROM for automatic booting from IPI disks, configuring a custom kernel for your SPARCserver 490, and configuring an Ethernet gateway.

2.2. Local Installation

Installation is divided into 12 major steps, with headings for each step in the left column. Note the chapter and section references in the left column beneath the heading for each major step; these references are to entries in *Installing the SunOS 4.0.3*. Refer to that document if you have any questions about details of the installation process.

Step 1: Gather Information and Plan Disk Space (Chapters 3 and 10 of *Installing the SunOS 4.0.3*)

You will need to know the following before you begin the installation; do not start the installation process unless you have this information. Most of the information can be obtained from your network or system administrator.

What you need to know:

- Device name of tape drive on which you will load release tapes (`st0`, `st1`, `st2` or `st3`). See Chapter 5 of this manual for instructions for determining the device name.
- Hostname of each machine (server and clients) you will be installing.
- Ethernet address of each machine you will be installing.
- Internet (IP) address of each machine you will be installing.
- Your domain name, if you use Sun's yp name service.
- Layout of partitions and file systems on disks.

If You Use yp

If your network uses the yp name service, be sure to add the new server and clients to the yp database before installing.

Step 2: Format, Partition, and Label Disks — if necessary (Sections 4.3 and 4.4)

Skip this step if you do not need to format, partition, or label your disks. Go to Step 3, "Load and Boot the miniroot."

Your disks are formatted and labeled before they are shipped by Sun. You may, however, want to format them again, or you may need to modify the default partitioning. If so, follow the instructions below, referring to Chapter 4 of *Installing the SunOS 4.0.3* if you have questions.

Note: munix (Memory UNIX) is a version of UNIX that uses a limited-function file system (munixfs) that resides in memory instead of on disk; munixfs includes the command `format`. Because you cannot format the disk that contains the root partition, you must run `format` from memory and not from disk.

1. **Halt the system, if necessary.**

   If the system is up and running, halt the system and bring it down to the boot PROM prompt (>) as gracefully as possible (with `/etc/halt`, for instance); use L1-a or Break, if necessary.

2. **Load the boot program into memory from the SPARCserver 490 tape.**

   Load the SPARCserver 490 tape into the appropriate tape drive and load the `boot` program into memory:

```
> bst()
```

   NOTE: Be sure you only have one tape mounted when you perform any of the booting steps; if a tape is mounted in your 1/4-inch drive while you are trying to install from your Front-Load tape drive (FLT), the tape in the FLT will not be found by the boot program. See Chapter 5 of this manual for an explanation of the device names (`st0, st1, st2, st3`).

   The `boot` prompt (Boot:) will be displayed.

```
Boot:
```

3. **Load and boot munix (Memory UNIX) from tape.**

```
Boot:  st(0,0,4)
Size:  594696+9232  (your size may vary)
SunOS Release 4.0.3 . . .
 . . .
 . . .
Initialize ram disk from device ( st%d[a-h] xt%d[a-h] id%3x[a-h] xy%d[a-h] sd%d[a-h] ft%d[a-h] ):
```

   The 4 in the above example — Boot: st(0,0,4) — represents the 4th file on the tape, which is munix; (see "Tape Contents" at the end of this chapter to see a table of contents for the SPARCserver 490 installation tape). There will be a slight delay as the file is found on the tape.

   The last line, beginning with "Initialize ram disk," comes from munix; it is asking which device you want to copy the munix file system from. All possible character-device controllers — `st, xt, id, xy, sd, ft` — are listed; your choice will be `st` (SCSI Tape). The pseudo-disk on which munix runs in memory is called a "ram disk."

4. **Specify where to copy the munix file system from:**

```
Initialize ram disk from device ( st%d[a-h] xt%d[a-h] id%3x[a-h] xy%d[a-h] sd%d[a-h] ft%d[a-h] ): st1
```

   It will usually be a tape drive, but munix allows you to copy from disk, also (thus the `xy`, `id`, and `sd` choices; `ft` is for booting over the network). Note that

"%d" is the C language convention for one decimal digit, and "%3x" is the convention for three hexadecimal characters. In the example above, `st1` is the device used.

You may have to substitute `st0`, `st2` or `st3` for `st1`; see Chapter 5 of this manual for an explanation of device names, if you are not sure what to use.

5. **Load munix file system from tape.**

   Specify where the munix file system, which includes the `format` program, is located on the tape (file 5):

```
Tape File #?   5
rd:  reading . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
2568192 bytes
root . . .
swap . . .
#
```

   When the # prompt appears, you are running single-user munix.

6. **Start the format program.**

   The available disks (0 and 1, in this example) will be listed, and you will be asked to select one:

```
# format
Searching for disks...done


AVAILABLE DISK SELECTIONS:
       0. id000 at id0 slave 0
          id000: <CDC IPI 9720>
       1. id001 at id0 slave 0
          id001: <CDC IPI 9720>

Specify disk (enter its number):
```

7. **Select the disk that you wish to format, partition, or label.**

   NOTE: you can only format disks on "slave" (controller) 0 from munix. If you have additional drives to format, do so after you have finished the installation and have booted vmunix from disk.

   In the next example, disk `id000` was selected; note line 3, which indicates that the disk is formatted and that a defect list was found. That line will not be displayed if the disk is not formatted. The `format` menu will be displayed:

```
Specify disk (enter its number): 0
selecting id000: <CDC IPI 9720>
[disk formatted, defect list found]

FORMAT MENU:
        disk       - select a disk
        type       - select (define) a disk type
        partition  - select (define) a partition table
        current    - describe the current disk
        format     - format and analyze the disk
        repair     - repair a defective sector
        show       - translate a disk address
        label      - write label to the disk
        analyze    - surface analysis
        defect     - defect list management
        backup     - search for backup labels
        quit
format>
```

8. **Format, partition, and label disk 0, if necessary.**

   See Section 4.4 of *Installing the SunOS 4.0.3* for information about formatting, partitioning, and labeling disks.

Saving Defect Lists

You should save defect lists to tape, in case the list is damaged on the disk. See the instructions for saving defect lists in Section 4.4 of *Installing the SunOS 4.0.3*.

9. **Quit format.**

   Type the letter q to quit `format`.

10. **Halt the system.**

    Halt the system and bring it down to the boot PROM prompt (>) with L1-a or Break.

Step 3: Load and Boot the miniroot (Sections 4.5 and 4.6)

To run SunInstall, you have to boot the miniroot from the SPARCserver 490 tape.

1. **Halt the system, if necessary.**

   If the system is up, halt the system and bring it down to the boot PROM prompt (>) as gracefully as possible (with `/etc/halt`, for instance); use L1-a or Break, if necessary (if you've just formatted your disks, the system will already be halted).

2. **Load the boot program into memory from the SPARCserver 490 tape.**

   Load the SPARCserver 490 tape into the appropriate tape drive and load the `boot` program into memory, as shown in the box below.

   NOTE: Be sure you only have one tape mounted when you perform any of

the booting steps; if a tape is mounted in your 1/4-inch drive while you are trying to install from your Front-Load tape drive (FLT), the tape in the FLT will not be found by the boot program. See Chapter 5 of this manual for an explanation of the device names (`st0, st1, st2, st3`).

```
> bst()
```

The boot prompt (Boot:) will be displayed.

3. **Enter st(0,0,2):**

```
Boot:  st(0,0,2)
Size: 21864+5536+82768 bytes
Standalone Copy
From:
```

This loads the Standalone Copy program into memory (Standalone Copy, which is needed to copy files from tape to disk, is file 2 on the tape; see "Tape Contents" at the end of this chapter to see a table of contents for the SPARCserver 490 installation tape).

4. **Specify miniroot (file 3 on tape) as the file to copy:**

```
From: st(0,0,3)
```

5. **Specify where on disk to copy miniroot:**

```
To:  id(0,0,1)
Copy completed — 6246400 bytes  (size may vary)
Boot:
```

Do not copy miniroot to partition "a"

Do not copy the miniroot to partition "a" of a disk; this will destroy the disk label.

In the above example, the miniroot is copied to partition "b" on the first system disk; note that "id" is the device abbreviation code for IPI disks. See Chapter 4 for a discussion of disk addressing.

6. **Boot vmunix from the miniroot on the disk:**

```
Boot:  id(0,0,1)vmunix
Size: . . .
SunOS Release 4.0.3_SPARCserver490 (GENERIC) . . .
 . . .
 . . .
root on id000b fstype 4.2
swap on id000b fstype spec size 28736K
dump on id000b fstype spec
#
```

When the # prompt appears, you are running single-user miniroot.

Step 4: Load SunOS 4.0.3 Installation Tape

Remove the SPARCserver 490 tape from the tape drive and insert the Sun-4 SunOS 4.0.3 tape into the drive. Note that if you do not load the SunOS 4.0.3 tape now, you will be told to load it later, when you are filling out the Software Form.

Step 5: Start suninstall (Chapter 5)

Type `suninstall` to start the installation process; enter your time zone and terminal type in response to system prompts:

```
#  suninstall

Welcome to SunInstall

You are about to install a new version of the SunOS on
 . . .
 . . .

Do you wish to continue with the installation [y/n]?  y

Enter the local time zone name:
>>  US/Pacific

Is this the correct date and time [y/n]:
>> Thu May 18 10:00:00 PST 1989

>>  y
```

(Type **n** *and enter the correct date and time if it is incorrect.)*

```
Select your terminal type:
    1) Televideo 925
    2) Wyse Model 50
    3) Sun Workstation
    4) Other
>> 3
```

The SunInstall Main Menu will be displayed:

```
                Sun Microsystems System Installation Tool

                                Main Menu

          On-line help information prints summary of cursor usage
                     + means the data file(s) exist(s)

                        assign host information

                        assign disk information

                        assign software information

                        assign client information

                        start the installation

                        on-line help information

                        exit from suninstall

[x/X=select choice] [space=next choice] [^B/^P=backward] [^F/^N=forward]
```

Use the space bar to move from field to field within a category of choices. Use Control-f or Control-n to move forward and down, Control-b or Control-p to move backward and up. Type the letter **x** to select an option. Note that you do not have to press Return after you type **x** to select an option; **x** is a "hot" button.

When you fill out the forms, you are designating what will be done once you start the installation; nothing is installed until then. You can exit SunInstall at any time without losing what you have entered — **as long as you do not leave the miniroot**. If you exit SunInstall, you will have to re-enter your time zone and terminal type when you start it up again.

Step 6: Complete Host Form (Chapter 5)

1. **Display the Host Form.**

   Move the cursor to the "assign host information" line and type **x** to display the form.

2. **Complete the Host Form.**

   See Section 5.7 of *Installing the SunOS 4.0.3* for detailed instructions for the Host Form, which asks you for workstation and network information.

If You Have a Second Ethernet

If you have a second Ethernet port, its device name (ie2) will be included on the Host Form, following the name of the primary Ethernet port (ie0). After filling out the IP address of the primary port, move the cursor to the next line and fill out the IP address of the second port. You *must* do this or you will not be able to boot the system to multi-user mode.

When you run SunInstall, entries for this second Ethernet port will be made in `/etc/rc.local` and `/etc/hosts`. See "Configure an Ethernet Gateway" in "Post-Installation Procedures," Section 2.3, below.

3. **Type y when you have completed the Host Form.**

```
Are you finished with this form [y/n] ? y
```

Note the plus sign (+) next to "assign host information" on the Main Menu, indicating that you have completed the form.

Step 7: Complete Disk Form (Chapter 5)

1. **Display the Disk Form.**

   Move the cursor to the "assign disk information" line and type **x** to display the form.

2. **Complete the Disk Form for first drive.**

   See Section 5.8 in *Installing the SunOS 4.0.3* for detailed instructions for the Disk Form, which lists every attached disk device and gives you the opportunity to modify the partitions on each disk, to designate a mount point for a partition, and to designate whether or not you wish to preserve an existing file system on a partition.

Changing root and swap Partitions

Do not change the size of the root partition and do not *reduce* the size of the swap partition on the Disk Form; if you do, the installation will fail. Such changes must be done with the `format` program, run from munix — *not* from the miniroot.

The /export Partitions

Read Chapter 8, "Advanced Installation Issues," in *Installing the SunOS 4.0.3*, for a discussion of the `/export` file system and what you need to do when installing diskless clients.

Freehog Partition for Heterogeneous Installations

Because SunInstall allocates excessive space for `/export/swap` in a heterogeneous installation, there may not be enough space in the `/usr` partition, which is where many of the software categories chosen on the Software Form are copied. When more space is needed for a category selected on the Software Form, the space is taken from the freehog partition. You need to increase the size of the freehog partition.

The easiest way to solve this problem is to reduce the size of the `/export/swap` partition — to the sum of the client swap partitions plus 10% (or whatever value you choose).

To reduce the size of the `/export/swap` partition, move the cursor to the SIZE field of the `/export/swap` partition, delete the default value, and type in a smaller number. As you move the cursor down to the next line on the form, you will notice that partition "h," the freehog partition, increases by the amount that the the `/export/swap` partition was decreased.

About the Preserve Option

Do *not* select Preserve if you are doing a first-time installation, or if the designated file system does not exist; the system will not boot successfully.

The Preserve option on the Disk Form allows you to save existing data on a disk partition. Be careful about using this option, since data may be lost if it is used improperly. Make note of the starting point and size of all partitions before making any changes; for partitions to be preserved, the starting

cylinder and size of the partition must remain the same or data may be lost. Remember that the freehog partition is automatically adjusted to accommodate changes to other partitions.

Data preservation may also be affected by software selections on the Software Form. If the selected categories will not fit onto the targeted partition (`/usr`), space will be taken from the freehog partition to satisfy your request. This may change the starting cylinder or size of other partitions, thus causing loss of data.

3. **Type y when you have completed the form for the first disk.**

```
Ok to use this partition table [y/n] ? y
```

4. **Complete the Disk Form for additional drives, if necessary.**

```
Ok to use this partition table [y/n] ? y
```

5. **Type y when you have completed the Disk Form for all disks.**

```
Are you finished with this form [y/n] ? y
```

If You Don't Assign All Disks

Note that no plus sign appears next to "assign disk information" until you have selected every available disk and completed the form for each. This is fine, since you may not wish to assign all disks at this time. Be aware that you may have to perform some system administration tasks later, though: formatting and partitioning disks, creating and mounting file systems, editing `fstab` to make file system mounting automatic during bootup, etc. — tasks otherwise performed for you by SunInstall.

Step 8: Complete Software Form (Chapter 5)

Before completing the Software Form, make sure the Sun-4 SunOS 4.0.3 tape is loaded in your tape drive. If it is not, SunInstall will tell you to load it as you fill out the form. The software categories you select on the Software Form will be extracted from the SunOS 4.0.3 tape; see "Tape Contents" at the end of this chapter for a list of the categories on the tape.

1. **Display the Software Form.**

   Move the cursor to the "assign software information" line and type **x** to display the form.

2. **Complete the Software Form for primary architecture (Sun-4).**

   When you select the Software Form, "sun4" is automatically selected, since the SPARCserver 490 is a Sun-4 machine; you must complete the Software Form for Sun-4 first.

   See Section 5.9 of *Installing the SunOS 4.0.3* for detailed instructions for the Software Form, which is where you tell SunInstall what type of machine architecture(s) you are installing, where the executables for a machine

reside, what type of tape drive will be used for the installation, and what software categories will be copied from the installation tape to the machine you are installing.

Tape Device Error Message

If you specify the wrong tape device (`st1` instead of `st0`, for instance) or if you select "local" or "remote" incorrectly, the following error message will be displayed at the bottom of the form:

```
mount a sun4 4.0.3 release tape
```

If you see this message and you are sure you have the proper tape in the drive, check your selections on the Software Form and correct them, if necessary.

3. **Respond to "OK to use the extract list" question.**

After you have completed the Software Form for Sun-4 architecture, the cursor will be at the bottom of the form, following the question "OK to use the extract list [y/n] ?" Respond **y** if you are satisfied with the list, **n** if you are not satisfied (you will have to select your choices again if you respond **n**).

When you respond **y** to the extract list question, the cursor will jump to the next line, "Are you finished with this form [y/n] ?"

If You Will Build a Customized Kernel

If you plan to build your own kernel, which is highly recommended, you must install the `Sys` category from the installation tape; otherwise, no kernel configuration files will be installed in `/sys`.

About the usr Partition

If you select more software categories than will fit in the `/usr` partition, space will be taken from the freehog partition. If there is not enough space in the freehog partition, you will have to go back to the Disk Form to increase the size of the freehog partition; the easiest way to increase the size of the freehog partition is to decrease the size of the `/export/swap` partition, as explained in the description of the Disk Form above.

Note also that you may want to go back to the Disk Form to add five to ten megabytes to the `/usr` partition to allow for customizing your kernel.

If You Use SunDials

If you use or plan to use the SunDials dialbox, note that the software for this product requires that certain SunView header files be loaded on your system in order that the product's test programs compile. Therefore, when you install your system, select the "SunView_Programmers" option as one of the categories to be installed.

4. **Respond to "Are you finished with this form?" question.**

Type **n** in response to the question at the bottom of the form if you have additional client architectures to install:

```
Are you finished with this form [y/n] ? n
```

Other Architectures — Sun-4c Last

5. **Complete the Software Form for other architectures, if necessary.**

   If your SPARCserver 490 is going to serve Sun-3, Sun-3x, or Sun-4c (SPARCstation 1) clients, you will have to complete the Software Form for each architecture. As you select an architecture on the form, you will be told to mount the SunOS 4.0.3 tape for that architecture.

   You need to complete the Software Form for each architecture that you will be installing. If you are installing only Sun-4 architecture, you need to fill out the form once only. If you are going to install other architectures, **you must select "sun4c" last** (if there are any SPARCstation 1 clients); during the installation, tapes will be requested in the order you select the architectures, and **Sun-4c must be installed last.**

6. **Type y when you have completed the Software Form for all architectures.**

```
Are you finished with this form [y/n] ? y
```

   Note the plus sign (+) next to "assign software information" on the Main Menu, indicating that you have completed the form.

Step 9: Complete Client Form — if necessary (Chapter 5)

If you have specified on the Host Form that your SPARCserver 490 will act as a server, you must complete the Client Form for each client.

1. **Display the Client Form.**

   Move the cursor to the "assign client information" line and type **x** to display the form — if you have client machines to install.

2. **Complete the Client Form for the first client to be installed.**

   See Section 5.10 of *Installing the SunOS 4.0.3* for detailed instructions for the Client Form, which needs to be completed only if you will be installing clients.

3. **Type n if you have other clients to install.**

   Type **n** in response to the question at the bottom of the form if you have additional clients to install:

```
Are you finished with this form [y/n] ? n
```

4. **Complete the Client Form for other architectures and clients, if necessary.**

5. **Type y when you have completed the Client Form.**

```
Are you finished with this form [y/n] ? y
```

   Note the plus sign (+) next to "assign client information" on the Main Menu, indicating that you have completed the form.

Step 10: Start the Installation (Chapter 5)

Select "start the installation" from the Main Menu and follow the instructions displayed on the screen; if you are installing more than one architecture, you will be asked to load the tapes for each architecture, once the previous architecture has been installed. You will be told when the installation for each architecture is complete.

Depending on what software categories you selected and what kind of tape drive you have, the installation will take between five and 30 minutes per architecture. When the entire installation is complete, the following message will be displayed:

```
4.0.3. installation completed.
Please reload SPARCserver 490 tape and run sunupgrade
to continue the installation procedure.
#
```

Step 11: Load SPARCserver 490 tape

Remove the SunOS 4.0.3 tape and reload the SPARCserver 490 tape.

Step 12: Run sunupgrade (Chapter 9)

Once you have installed SunOS 4.0.3, you are ready to run `sunupgrade`, which will copy SPARCserver 490 files from tape to disk.

```
# sunupgrade
```

See Chapter 9 in *Installing the SunOS 4.0.3* for a detailed description of the upgrade process. In general, `sunupgrade` is simple to follow: just respond to system prompts reasonably. See an example of some of the system prompts and responses below.

```
# sunupgrade
Enter root disk partition for sun4 architecture (e.g. xy0a): id000a
Wait . . (This may take a while – up to three minutes.)
Is this a file server (as opposed to a standalone/dataless client)? (y/n) y
WARNING! If you are not sure of any of the answers to
    the questions below, exit the miniroot, come up
    in the multi-user mode, obtain necessary information
    and restart the whole process.  Refer to the
    Read This First documentation for details.
The following assumptions are being made for this upgrade:
1. You have not changed your file system from the Sun standard.
2. If Sunupgrade finds that this is a server without any client
   root partitions (eg. A server serving dataless clients only),
   it will upgrade only the client user areas.  If yours is such
   a case, it does not matter what your answer in response to the
   query on client root partition name.
3. If this is a server, the client partitions are assumed to
   be as follows:

    All client usr file partitions are under: /export/exec
    All client root file partitions are under: /export/root
    All client swap file partitions are under: /export/swap
 Hit return to continue.
4. All client usr partitions MUST be in one directory.
   Similarly for client root partitions (if they exist).
   If you have client usr (or root) partitions scattered in
   more than one location, exit now and do the following:
   Select one of the directories containing the most number
   of exec directories as your primary exec directory, and one of
   the client root directories as your primary client root
   directory.  Set up symbolic links bearing the same names
   (with relative pathnames) in your primary directory, to the
   entries wherever they may be , and restart sunupgrade.
   When you come to this menu, enter the primary directory names at
   the prompt.
Refer to Installation Instructions for details.
```

Continue the `sunupgrade` process. In the example below, the default locations for client `usr`, `root`, and `swap` partitions were accepted, the tape drive is local, and the installation tape is in drive `st0`.

Note that choice number 1, " Upgrade all," should be selected when you are asked if you want to upgrade all, upgrade server only, or upgrade clients only.

```
Continue? (y/n): y

Change clients usr file partition from /export/exec? n

Change clients root file partition from /export/root? n

Change clients swap file partition from /export/swap? n

Where is the tape drive located? [local | remote]? local

Enter controller type [st | mt | xt ]: st

Enter tape device address [0 | 1 | 2 | 3]: 0

Use st0 (y/n): y

Extracting TOC (Table of Contents)
0+1 records in
0+1 records out

1. Upgrade all
2. Upgrade Server only
3. Upgrade Clients for same architecture only

Select one of the above.  The default is 1.
Enter choice: 1

Starting upgrade now.  Continue? (y/n): y

This is going to take some time.

Extracting "root" files

Extracting "usr" files

Installing bootblock to root partition /dev/rid000a

Installing /sbin files.

Doing file system checks
/dev/rid000a: nnn files, nnn used, nnn free
 . . .
 . . .
sunupgrade:  Done upgrading to 4.0.3_SPARCserver_490.
```

When `sunupgrade` is complete, a series of possible next steps will be listed:

```
NEXT STEPS
    * Install small pre-configured kernel using install_small_kernel (OPTIONAL)
    * Reboot and come up single user
    * Check and install special_files (look in /usr/etc/upgrade/save)
    * Reconfigure your kernel
    * Come up in multi-user mode
#
```

2.3. Post-Installation Procedures

See Chapter 7 in *Installing the SunOS 4.0.3* for a list of post-installation steps, including instructions for configuring a customized kernel. See Sections 9.3 and 9.8 for information about `special_files`, where you can find out which files were *not* installed during `sunupgrade` because the file already existed on your system; you can look at both files and decide whether to replace the old one with the new one, or if you need to modify one or the other.

You Must Edit /etc/netmasks

Note that if you have a second Ethernet you must edit `/etc/netmasks` after you boot the system in single-user mode; see the instructions below.

Install Small Kernels

The utility `install_small_kernel` installs small pre-configured kernels on your SPARCserver 490 for client systems which are either diskless or equipped with SCSI devices. The kernel supports about four users for the following diskfull or diskless configurations:

Sun-2/50 with up to 2 SCSI disks, 1 SCSI tape
Sun-3/50 and Sun-3/60 with up to 2 SCSI disks, 1 SCSI tape
Sun-3/80 with up to 4 SCSI disks, 1 SCSI tape
Sun-4/110 with up to 2 SCSI disks, 1 SCSI tape
Sun-4/330 with up to 4 SCSI disks, 1 SCSI tape

When you run `install_small_kernel` you will be asked, for each client defined on the Client Form, whether or not you wish to use the small kernel for that machine's architecture. See Sections 7.5 and 7.6 in *Installing the SunOS 4.0.3* for detailed information about `install_small_kernel`.

NOTE: the script `install_small_kernel` can only be run immediately *after* you do a full install of SunOS 4.0.3 and an upgrade of SPARCserver 490 software, and *before* you quit the miniroot (files created by SunInstall and `sunupgrade` in the miniroot are necessary for `install_small_kernel`).

Halt from the Miniroot

Once you have run `install_small_kernel`, you are finished with the miniroot; to continue, halt the system with L1-a or Break.

Set EEPROM for IPI Booting

See Chapter 4 in this manual for instructions for setting the EEPROM to boot from a specific IPI disk drive. If the EEPROM is not explicitly set, the system may boot from an unexpected device, causing an unknown condition.

Boot System in Single-User Mode

Before bringing the system up in multi-user mode, you may want to customize your kernel, configure an Ethernet gateway, configure a yp master, or perform other activities described in Chapter 7 of *Installing the SunOS 4.0.3*. To boot in single-user mode:

```
>b -s
```

Customize Your Kernel

You will probably want to use something other than the GENERIC kernel, which is automatically installed during SunInstall. If you have many users on your SPARCserver 490, for example, you may want to increase the "maxusers" variable from its default value of 24.

Optionally, you may wish to use the smaller kernel (IDST490) provided for the SPARCserver 490; see Chapter 3 in this manual for a description of IDST490, instructions for using a kernel other than the GENERIC kernel, and suggestions for customizing a kernel.

Configure an Ethernet Gateway

If you completed an entry for your second Ethernet port on the Host Form, as required, SunInstall will have configured it as a network gateway; entries will have been made in `/etc/rc.local` and `/etc/hosts`.

You may wish to change those entries; in particular, SunInstall enters the gateway as 'hostname'-gw (the name of your SPARCserver 490 will automatically be substituted for "hostname"). If you want a different name for your gateway, modify the entry in `/etc/rc.local` and `/etc/hosts`.

See Chapter 12, "The SunOS Network Environment" in *System & Network Administration*, and the networks (5) man page for more information.

You Must Edit /etc/netmasks if You Have a Second Ethernet

Before you can boot the system in multi-user mode, you must edit the file `/etc/netmasks`, substituting the first two fields of the IP address of your SPARCserver 490 for the values there by default.

If your SPARCserver 490 is a yp client, make the change on the yp server; if your SPARCserver 490 is not a yp client, make the change on your SPARCserver 490.

If Your SPARCserver 490 Is a yp Server

Edit `/etc/netmasks` on your SPARCserver 490 to reflect your internet address.

Suppose the internet address of your SPARCserver 490 is 129.144.60.4. By default, `/etc/netmasks` has the following entry:

```
# Network       netmask
128.32.0.0      255.255.255.0
```

Change this to:

```
# Network       netmask
129.144.0.0     255.255.255.0
```

Note that only the first two fields of the Network number need to be changed.

Setting yp Maps for yp Server

If your SPARCserver 490 is a yp server, be sure to complete the following steps before you attempt to boot the clients. Remember that you cannot boot your SPARCserver 490 if you have not edited `/etc/netmasks` as described above.

1. **Run /usr/etc/yp/ypinit.**

```
# ypinit -m
```

Run ypinit with the -m argument to set up maps; see the *System and Network Administration Manual* for details about ypinit.

2. **Edit /etc/rc.local.**

As root, edit the `/etc/rc.local` file and remove the # signs from the code that tests for ypbind:

```
#if [ -f /usr/etc/ypbind ]; then
#    if [ -f /etc/security/passwd.adjunct ]; then
#       ypbind -s;    (echo -n ' ypbind')    >/dev/console
#    else
#       ypbind;       (echo -n ' ypbind')    >/dev/console
#    fi
#fi
```

3. **Execute ypbind or reboot the server.**

If Your SPARCserver 490 Is a yp Client

If your SPARCserver 490 is a yp client, make sure your yp server has the correct netmask entry. Follow the above procedure to change the yp server netmask entry and propagate the new netmask map on the network by issuing the following commands on your yp server.

```
# cd /var/yp
# make netmasks
```

Boot System Multi-User

Once you have completed all the installation and post-installation procedures in single-user mode, bring the system up in multi-user mode (CTRL -d if in single-user mode; b Return if at > prompt).

2.4. Dataless Client Installation

If you will be installing dataless clients of your SPARCserver 490, perform the installation as a separate procedure from the SPARCserver 490 installation, using the appropriate SunOS 4.0.3 installation tapes. See Section 6.6, "Local Installation: Dataless" in *Installing the SunOS 4.0.3*. Note that you need to specify only root and swap for the client's local disk on the Disk Form; other software is mounted from the server.

You will have to install the appropriate architecture for a dataless client, just as for any other SPARCserver 490 client; if your dataless client is a Sun-3, for

instance, be sure to install Sun-3 architecture as part of the SPARCserver 490 installation.

If you decide to add a dataless client to your SPARCserver 490 after you have completed the installation on your SPARCserver 490, and if the client's architecture has not been installed on the SPARCserver 490, you will have to add the client architecture, as described in Section 2.5, "Adding Support for Other Architectures After Installation."

2.5. Adding Support for Other Architectures After Installation

Normally, you will add support for all architectures during SunInstall, as described in Section 2.2 above. If you do not add the support at that time, however, you can do it later, after the system is up and running.

To add software support for new client architectures after you have completed the SPARCserver 490 software installation, do the following on your SPARCserver 490. See Section 8.3 of "Advanced Installation Issues" in *Installing the SunOS 4.0.3* for descriptions of `setup_exec` and `setup_client`. The six steps of the procedure, which are described in detail below, are:

1. Run `setup_exec`
2. Halt the system.
3. Load the miniroot from the SPARCserver 490 tape.
4. Run `sunupgrade` (upgrade server only).
5. Boot the system in multi-user mode.
6. Run `setup_client` to add clients.

1. **Run setup_exec.**

   In multi-user mode, as the superuser, run `setup_exec` (`/usr/etc/install/setup_exec`). Type `setup_exec` with no arguments to see a usage line that includes the proper format of the `arch` argument for each supported architecture. To add Sun-3 architecture, for instance, type the following:

```
# setup_exec sun3 /export/exec/sun3 /export/exec/kvm/sun3
```

   When you run `setup_exec` the Software Form from SunInstall is displayed. Complete the form for each architecture to be added, responding **n** when asked if you are finished with the form. When you have completed the form for all architectures (Sun-4c last), respond **y** to the "Are you finished with this form?" question; the installation will begin.

If You Add Sun-4c and Other Architectures Together

You must install the software for Sun-4c architecture last; if you are adding Sun-4c and other architectures, complete the Software Form for the Sun-4c last. Note that the Sun-4c (SPARCstation 1) encryption kit is not needed when installing clients of a SPARCserver 490 server, since files in the kit are contained in the 4.0.3 domestic tape set for the Sun-4, installed as part of the SPARCserver 490 installation.

Reinstall Sun-4c if You Add Other Architectures Later

If you have *previously* installed Sun-4c architecture and you add another architecture (Sun-3, Sun-3x) with `setup_exec`, you must *reinstall* Sun-4c architecture (using `setup_exec`) before halting the system and running `sunupgrade`. This is necessary because of Sun-4c architecture dependencies which are written over by the installation of the other software.

Three categories of software must be reinstalled when you reinstall Sun-4c architecture after adding other architectures:

Kvm
Sun-4c_Usr
Sys

When you run `setup_exec` to reinstall the Sun-4c, include the -o option, which tells `setup_exec` to override software protection.

```
# setup_exec sun4c /export/exec/sun4c /export/exec/kvm/sun4c -o
```

Respond **n** if you see the message:

Do you want to use existing software list [y/n] ?

Select "[own choice]" on the Software Form and respond "y" to the three categories listed above.

```
Choice  :  [all]  [default]  x[own choice]  [required]  [quit]
```

2. **Halt the system.**
3. **Load the miniroot from the SPARCserver 490 tape.**

   See Step 3 in "Local Installation" (Section 2.2 above) for instructions. Start with number 2 in the procedure, "Load the boot program into memory from the SPARCserver 490 tape," and go through number 6, "Boot vmunix from the miniroot on the disk."
4. **Run sunupgrade.**

   Select number 2, "Upgrade Server only," when you are asked if you want to upgrade all, upgrade server only, or upgrade clients only. See Step 12 in the local installation procedure (Section 2.2 above) for detailed instructions and examples.
5. **Boot system multi-user.**
6. **Run setup_client to add individual clients — once for each client to be added.**

Reverting to a Previous Version of vmunix

Note that `sunupgrade` writes over `/vmunix` and saves it to `/vmunix-pre.4.0.3.SPARCserver.490`. If you want to go back to your original vmunix, bring the system up in single-user mode, copy the saved `vmunix` (`/vmunix-pre.4.0.3.SPARCserver.490`) to `/vmunix`, and reboot.

2.6. Remote Installation

Follow the instructions below to install SPARCserver 490 software on a SPARCserver 490 that does not have its own 1/4-inch or 1/2-inch tape drive. A remote installation is performed from a Sun-3 or Sun-4 with a 1/4-inch or 1/2-inch tape drive. This procedure is similar, but not identical, to the procedure described in *Installing the SunOS 4.0.3*, "Remote Installation: Standalone"; refer to that manual if you have general questions about remote installations, but follow the steps described here.

The machine from which the installation is performed is called a *remotehost*. The *remotehost* is used to boot the SPARCserver 490, using the miniroot from the SPARCserver 490 installation tape.

Requirements for performing a remote installation

Note the following requirements for performing a remote installation:

- The *remotehost* must be running SunOS 4.0.3.
- The script `setup_client` must exist on the *remotehost*. This script will exist in `/usr/etc/install/script` if you used SunInstall to install SunOS 4.0.3.
- The *remotehost* must have Sun-4 binaries. Use `setup_exec` to load the Sun-4 binaries if they are not available on your Sun-3 (they *are* available on your Sun-4 machine). See Section 8.3 of "Advanced Installation Issues" in *Installing the SunOS 4.0.3* for a description of `setup_exec`.
- A 'spare' disk partition must be available on the *remotehost* — at least seven megabytes in size, to contain the miniroot and the boot file from the SPARCserver 490 installation tape.
- You must know the hostname and the Ethernet and internet addresses of the SPARCserver 490 you are installing. If you are running the yp name service, these Ethernet and internet addresses must be entered on the yp master and propagated to *remotehost*. If you are not running yp, the Ethernet and internet addresses of the SPARCserver 490 must be entered in `/etc/hosts` and `/etc/ethers` on *remotehost*.
- The name of your SPARCserver 490 must be in `/.rhosts` on *remotehost*; create `/.rhosts` if it does not exist.

Step 1: Create Temporary Client on *remotehost*

This client will be removed once the remote installation is complete.

1. **Change directory to the location of setup_client on *remotehost*:**

```
% cd /usr/etc/install/script
```

2. **Run setup_client on *remotehost*, as root.**

   Type the entire command-line sequence before pressing Return:

```
# setup_client add sundust client 16m /export/root \
/export/swap /home /export/exec/sun4 /export/exec/kvm/sun4 sun4
Start creating sun4 client "sundust":
Updating bootparams ...
ATTENTION: /etc/bootparams on the yp master needs to be updated
Creating root for client "sundust".
Creating 16m bytes of swap for client "sundust".
Updating /etc/exports to export "sundust" info.
  . . .
  . . .
Completed creating sun4 client "sundust".
```

   In the preceding example, we added the client "sundust," with the following characteristics:

   - sundust is a yp client
   - has a swap size of 16 megabytes
   - client root in `/export/root`
   - client swap located at `/export/swap`
   - client home located at `/home`
   - client exec path is `/export/exec/sun4`
   - client kvmpath is `/export/exec/kvm/sun4`
   - client architecture is Sun-4

   Note: enter all arguments before pressing Return. Type `setup_client` with no arguments to see help for `setup_client`. Run `setup_client` just as in the example above, substituting the name of your SPARCserver 490 for "sundust." If you do not have 16 megabytes of disk space available on *remotehost*, substitute a smaller number for swap size.

When `setup_client` has completed successfully, the following will have been created or added to, depending on whether or not you are running the yp name service:

If you are running yp:

- the yp map "bootparams" will have an entry.

  You must propagate it to *remotehost* (if *remotehost* is not the yp master).

If you are not running yp:

- `/etc/bootparams` will contain the pathname to the root directory tree (Sun default is `/export/root/`*`clientname`*).

In both cases:

- `/tftpboot` (a directory) will contain a file whose name is composed of the hexadecimal equivalent of the IP address of the SPARCserver 490.

  This file will be a symbolic link to file boot.sun4, e.g., 81903D9B.SUN4 -> boot.sun4

- a root file system, at the location specified in the bootparams entry.

  This file system will be replaced in the next step.

Step 2: Copy miniroot and boot file from SPARCserver 490 tape to the *remotehost*

Copy the miniroot and the boot file from the SPARCserver 490 tape to the 'spare' disk partition on *remotehost*. The disk partition will be referred to as `/dev/`*`xxxp`* in these instructions; substitute the device name (e.g., `/dev/sd2h`) of the 'spare' disk partition wherever `/dev/`*`xxxp`* is shown.

Warning: these activities can be dangerous; the procedure must be followed very carefully.

Be sure you have backed up the data from the disk partition, if there is any useful data on it; all data on the partition will be lost during this procedure. Also note that this disk partition must not be mounted and **must not begin on cylinder 0 of the disk**; if it does, this procedure will destroy the disk label and partition table. Finally, be careful in executing the following commands; as with any commands which directly use `/dev` names, an error can result in loss of all data on an entire disk drive. If you have doubts about these procedures, do not proceed.

Note that the instructions use the no-rewind, raw device name for the tape drive; be sure to precede the device name with "nr" in the following commands. Substitute the appropriate device name (`st0, st1, st2`, etc.) for {*tape*} in the commands.

1. **Unmount the 'spare' file system:**

```
# umount /dev/xxxp
```

2. **Copy the miniroot from the SPARCserver 490 tape:**

```
# mt -f /dev/nr{tape} rew
# mt -f /dev/nr{tape} fsf 3
# dd if=/dev/nr{tape}  of=/dev/xxxp bs=16k
```

The first `mt` command rewinds the tape to the beginning. The second mt command skips forward past three files, positioning the tape at the beginning of the fourth file, which is the miniroot. The `dd` command copies the miniroot to the disk partition, `/dev/`*xxxp* (substitute the 'spare' partition discussed above).

3. **Copy the boot file from the Kvm category on the tape.**

```
# cd /tmp
# mt -f /dev/nr{tape} fsf 4
# tar xvf /dev/nr{tape} stand/boot.sun4
```

In the above example, the file `stand/boot.sun4` is copied to `/tmp`; you can copy it to any mounted file system. The `mt` command skips forward four more files on the tape (the tape was not rewound after the previous commands), thus positioning the tape at the beginning of the ninth file on the tape, Kvm. The `tar` command extracts the file specified (`boot.sun4`) from the current tarfile and copies it to the current directory (`/tmp`), creating the directory `stand`, containing the file `boot.sun4`.

Step 3: Mount the miniroot on the root directory of your SPARCserver 490

By mounting the disk partition containing the miniroot on top of the root directory for your SPARCserver 490 client (created in Step 1 above), you make the miniroot accessible to the SPARCserver 490 client. The miniroot has the kernel and utilities needed by the SPARCserver 490 to perform the remote installation. Substitute the name of your SPARCserver 490 for "clientname" in the command below:

```
#   mount /dev/xxxp /export/root/clientname
```

Step 4: Export the new file system

Inform the NFS mount daemon there is a new file system to be accessed by the client SPARCserver 490.

```
# exportfs -a
```

Step 5: Determine which file in /tftpboot is your SPARCserver 490

The directory `/tftpboot` contains one file for each client created with `setup_client`. These files created by `setup_client`, have file names which are comprised of the hexadecimal equivalent of the IP address of the client, plus a suffix that represents the architecture of the client. If you have set up two Sun-4 clients, an `ls` in `/tftpboot` would show something like:

```
81903D9B.SUN4
81903D3C.SUN4
```

To determine which file name represents the SPARCserver 490 you are installing, do the following (assuming the name of the SPARCserver 490 is "corn"):

```
# ypmatch corn hosts (if you are running yp)
                    or
# grep corn /etc/hosts (if you are not running yp)
129.144.61.155 corn
# /usr/etc/install/script/convert_to_hex 129.144.61.155
81903D9B
```

So now you know that 81903D9B.SUN4 is the SPARCserver 490 you are installing.

Step 6: Point the network boot to the SPARCserver 490 boot program

In the box below, you copy the boot program from the SPARCserver 490 installation tape to /tftpboot, the network-boot directory, and link the hex-equivalent name of your SPARCserver 490 to the SPARCserver 490 boot program. This allows you to use the SPARCserver 490 boot program when you boot over the network from your SPARCserver 490.

```
# cd /tftpboot
# cp /tmp/stand/boot.sun4 boot.490
# rm 81903D9B.SUN4   (removes link to boot.sun4)
  (Substitute hex name of your SPARCserver 490 + .SUN4)
# ln -s boot.490 81903D9B.SUN4
  (Substitute hex name of your SPARCserver 490 + .SUN4)
```

Step 7: Do a network boot from the SPARCserver 490

Go to your SPARCserver 490 and boot it over the network (halt the system if it is up and running):

```
>b ie() -a
Using IP address 129.144.19.26 (IP address of the SPARCserver 490)
Booting from tftp server at 129.144.19.2 (IP address of remotehost)
Downloaded nn bytes.  (some number)
root filesystem type (..):  nfs
Using IP address 129.144.19.26 (IP address of the SPARCserver 490)
hostname:  server-name  (the name of your SPARCserver 490)
domainname:  xxx  (your domain name, if any)
rootname:  <Return>
server name:  xxx            (the name of your remotehost)
root pathname:  xxx          (entry from bootparams)
root on server:  /xxx
Boot: vmunix
root filesystem type (...): nfs
rootname: <Return>
swap filesystem type (...): nfs
swapname: <Return>
#
```

Your SPARCserver 490 is running the miniroot. You can now format your disks, if desired, and run SunInstall, as described in 2.2, "Local Installation," above. You will specify "remote" for "Drive Type" on the Software Form and fill in the necessary address for *remotehost*.

Step 8: Run SunInstall

Load the Sun-4 SunOS 4.0.3 installation tape in the tape drive on the *remotehost* and invoke SunInstall from your SPARCserver 490:

```
# suninstall
```

See Section 2.2 above for more information about the installation process. Note that remote installations tend to take longer than local installations.

When SunInstall is complete, you will be asked to mount the SPARCserver 490 tape and run `sunupgrade`, just as in a local installation.

Step 9: Run sunupgrade

```
# sunupgrade
```

When the upgrade is complete, you will see a list of post-installation steps, just as you did for a local installation. See Section 2.3 above for information about these steps:

```
NEXT STEPS
    * Install small pre-configured kernel using install_small_kernel (OPTIONAL)
    * Reboot and come up single user
    * Check and install special_files (look in /usr/etc/upgrade/save)
    * Reconfigure your kernel
    * Come up in multi-user mode
#
```

Step 10: Restore the spare disk partition on *remotehost*

1. **Unmount the partition:**

```
# umount /dev/xxxp
```

2. **Run newfs to return the partition to its normal size:**

```
#   newfs /dev/rxxxp (e.g., rsd2h)
```

Note that you need to run `newfs` on the raw device: include the **r** in the device name.

3. **Restore the original file system if you backed it up prior to the remote installation.**

Step 11: Remove files created on *remotehost*

```
# rm /tftpboot/boot.490 /tftpboot/xxxxxxxx.SUN4
```

Step 12: Remove the SPARCserver 490 client from *remotehost*

Run `setup_client` as before, but substitute "remove" for "add" (type the entire command-line sequence before pressing Return):

```
# setup_client remove sundust client 16m /export/root \
/export/swap /home /export/exec/sun4 /export/exec/kvm/sun4 sun4
```

2.7. Tape Contents

When you complete the Software Form you tell SunInstall which software categories you wish to extract from the SunOS 4.0.3 installation tape for each architecture. This section of the manual tells you which categories are on the SunOS 4.0.3 tape, and which categories are on the SPARCserver 490 tape.

One of the files on an installation or upgrade tape is the table of contents for the tape; the file is named XDRTOC. Shown below are the tables of contents for a SunOS 4.0.3 installation tape and for the SPARCserver 490 installation tape; following the tables of contents are instructions for reading a table of contents from a tape and instructions for extracting a list of the individual files within a software category on a tape.

SunOS 4.0.3 Installation Tape Table of Contents

The SunOS 4.0.3 Installation Tape (Sun-4) should contain the following software categories.

```
SunOS 4.0.3 700-2165-10 from Sun Release Engineering.
ARCH sun4
VOLUME 1
 Vol File              Name       Size   Type
   1    0              boot      49152   image
   1    1            XDRTOC       4096   toc
   1    2              copy      49664   image
   1    3         mini-root    6154240   image
   1    4             munix    1016320   image
   1    5           munixfs    2105344   image
   1    6              root      51200   tar
   1    7               usr   16281600   tar
   1    8               Kvm    3338240   tar
   1    9           Install     870400   tar
   1   10               Sys    1925120   tar
   1   11        Networking    1024000   tar
   1   12         Debugging    4218880   tar
   1   13     SunView_Users    1249280   tar
   1   14 SunView_Programmers 1454080   tar
   1   15      SunView_Demo      61440   tar
   1   16              Text     624640   tar
   1   17         User_Diag    4126720   tar
   1   18           SunCore    1679360   tar
   1   19              uucp     286720   tar
   1   20          System_V    4372480   tar
   1   21            Manual    6338560   tar
   1   22              Demo     245760   tar
   1   23             Games    2734080   tar
   1   24          Versatec     102400   tar
   1   25          Security     184320   tar
   1   26         Copyright       1024   image
```

SPARCserver 490 Installation Tape Table of Contents

As of the time this document went to press, the SPARCserver 490 installation tape contained the following software categories. See the instructions following this table of contents for a method for extracting the actual table of contents, which may be different.

```
SunOS 4.0.3_SPARCserver_490 700-2481-10 Rev. A of [date & time]\
from Sun Release Engineering.
ARCH sun4
VOLUME -1
 Vol File              Name       Size   Type
   1    0              boot      49152   image
   1    1            XDRTOC       4096   toc
   1    2              copy      49664   image
   1    3         mini-root    6246400   image
   1    4             munix    1065472   image
   1    5           munixfs    2150400   image
   1    6              root     102400   tar
   1    7               usr     819200   tar
   1    8               Kvm    2867200   tar
   1    9           Install     819200   tar
   1   10               Sys    1433600   tar
   1   11         User_Diag    2252800   tar
   1   12            Manual     204800   tar
   1   13         Copyright       1024   image
```

Extracting the Table of Contents from a Tape

Do the following to extract the table of contents from an installation or upgrade tape. You may wish to do this to confirm that the software categories listed in the tables of contents above are accurate.

Log on as root, mount the tape in your tape drive, and execute the following commands to generate a table of contents from the tape (example below is for 1/4-inch tape, mounted on `st0`; replace **`rst0`** with **`rst1`**, **`rst2`**, or **`rst3`** if the tape is in one of those drives):

```
# mt -f /dev/nrst0 asf 1
# dd if=/dev/nrst0 | /usr/etc/install/xdrtoc
6+0 records in
6+0 records out
SunOS 4.0.3_SPARCserver_490 700-2481-10 Rev. A of [date & time]\
from Sun Release Engineering.
ARCH sun4
VOLUME -1
 Vol File              Name       Size   Type
   1    0              boot      49152   image
   1    1            XDRTOC       4096   toc
   1    2              copy      49664   image
   1    3         mini-root    6246400   image
   1    4             munix    1065472   image
   1    5           munixfs    2150400   image
   1    6              root     102400   tar
   1    7               usr     819200   tar
   1    8               Kvm    2867200   tar
   1    9           Install     819200   tar
   1   10               Sys    1433600   tar
   1   11         User_Diag    2252800   tar
   1   12            Manual     204800   tar
   1   13         Copyright       1024   image
#
```

(The `mt` command finds file 1 on the tape. The `dd` command reads from the current location on tape to a file mark. The `xdrtoc` command translates the table of contents into human-readable format and displays the result on the screen.)

Listing Files in a Software Category

To list the individual files that comprise a software category, use the following procedure:

1. Mount the distribution tape which contains the desired software category (the SunOS 4.0.3 installation tape, in the example below).
2. Skip to the correct file (the "uucp" category, which is file 19 on the tape, as seen in the Table of Contents on a previous page).
3. List the table of contents for the current category with the "tar tvf" command.

   To see the files included in the "uucp" category on the SunOS 4.0.3 installation tape, for example, assuming a 1/4-inch tape:

```
# mt -f /dev/nrst0 asf 19
# tar tvf /dev/nrst0

rwxrwxr-x  4/10      0 Feb  9 14:46 1989 ./lib/uucp/
rwx------  4/10    648 Feb  3 14:47 1989 ./lib/uucp/uuck
--x------  4/10  16384 Feb  3 14:47 1989 ./lib/uucp/uusub
--s--x--x  4/10  32768 Feb  3 14:47 1989 ./lib/uucp/uuxqt
#
```

   Substitute **nrst1**, **nrst2**, or **nrst3** if the tape is in drive `st1`, `st2`, or `st3`, respectively.

3

Kernel Configuration

Kernel Configuration **45**

3.1. Supplied Kernel Configuration Files 45
The GENERIC Kernel 45
The SPARCserver 490 Kernel: IDST490 45
3.2. Customizing a Kernel 50
Create /dev Entries 50
Make Entries in /etc/ttytab 51
Possible Error Messages 52
Monitoring Performance 52
3.3. Configuring a Kernel 53

3

Kernel Configuration

Because the SPARCserver 490 is a unique hardware configuration, with many variables, you will probably want to customize your kernel so as to optimize performance. Kernel configuration options are briefly covered below, and are fully discussed in *Installing the SunOS 4.0.3* (Chapter 7).

Section 3.1 below describes the GENERIC and IDST490 kernels; Section 3.2 explains how to customize a kernel. Read both sections before attempting to customize a kernel.

3.1. Supplied Kernel Configuration Files

A number of kernel configuration files are supplied in the directory `/usr/sys/sun4/conf`, including GENERIC and IDST490, which are the two kernels of interest for your SPARCserver 490. Do not attempt to use GENERIC_SMALL, which is for the Sun-4/110 or the Sun-4/330.

The GENERIC Kernel

After a full SPARCserver 490 installation, a large GENERIC kernel has been installed, by default. The GENERIC kernel contains all available device drivers and options for all Sun systems, including many drivers and options that you might not need. As noted in the installation instructions (Chapter 2 above), if you have many users on your SPARCserver 490, you may want to increase "maxusers" from its default value of 24. Note that the GENERIC configuration file is heavily commented for easy reading.

The SPARCserver 490 Kernel: IDST490

The IDST490 kernel is a model small SPARCserver 490 configuration; if IDST490 does not have all that you need, and GENERIC has more than you need, it will probably be easier and safer to add entries to IDST490 rather than deleting entries from GENERIC. Use the GENERIC kernel as a guide for the format of entries you add to IDST490.

The IDST490 kernel has a `maxusers` value of 24 and is tailored for a SPARCserver 490 with four ISP-80 controllers, 32 IPI8-1000 disk drives, and two SCSI tape drives on the first SCSI bus; add or subtract devices and modify variables, according to your system's configuration.

Shown on this page and the following three pages is a listing of the IDST490 kernel configuration file, which is included on the SPARCserver 490 installation tape, and which is installed in the directory `/usr/sys/sun4/conf` — *if* you choose the software category `Sys` on the Software Form during SunInstall.

Note the following new entries in the configuration file:

- IPI entries: all entries that include "ipi" or that are preceded by comments that include "IPI" are related to the new ISP-80 disk controller or the IPI8-1000 disk drives. This includes controller entries `isc0, isc1, isc2,` and `isc3`; channel entries `is0, is1, is2,` and `is3`; controller entries `idc0, idc1, idc2,` and `idc3`; disk entries `id0 - id7, id0x10 - 0x17, id0x20 - id0x27,` and `id0x30 - id0x37`.
- Ethernet entry: the second Ethernet port, which can be used as a network gateway, is `ie2`.

```
# @(#)IDST490 1.1 89/09/07 SMI
#
# This config file describes a sparcstation Sun-4/490 kernel
machine     "sun4"
cpu             "SUN4_460"      # Sun-4/460 or Sun4/490
ident       "IDST490"
maxusers    24
options     INET        # basic networking support - mandatory
options     QUOTA       # disk quotas for local disks
options     UFS         # filesystem code for local disks
options     NFSCLIENT   # NFS client side code
options     NFSSERVER   # NFS server side code
#options    LOFS        # loopback filesystem - needed by NSE
options     SYSACCT     # process accounting, see acct(2) & sa(8)
#options    SYSAUDIT    # C2 auditing for security
options     IPCMESSAGE  # System V IPC message facility
options     IPCSEMAPHORE    # System V IPC semaphore facility
options     IPCSHMEM    # System V IPC shared-memory facility
#options    TCPDEBUG    # TCP debugging, see trpt(8)
options     CRYPT       # software encryption (if no DES chip)

# Put the kernel configured this way in a file named "vmunix".
#
config      vmunix      root on id

#
# Include support for all possible pseudo-devices.
#
# The first few are mostly concerned with networking.
# You should probably always leave these in.
#
pseudo-device   pty     # pseudo-tty's, also needed for SunView
pseudo-device   ether       # basic Ethernet support
pseudo-device   loop        # loopback network - mandatory
pseudo-device   win128      # window devices, allow 128 windows
pseudo-device   dtop4       # desktops (screens), allow 4
pseudo-device   ms3     # mouse support, allow 3 mice
```

(continued on next page)

(continued from previous page)

```
#
# The following is needed to support the Sun keyboard, with or
# without the window system.
#
pseudo-device   kb3     # keyboard support, allow 3 keyboards
pseudo-device   snit        # streams NIT
pseudo-device   pf      # packet filter
pseudo-device   nbuf        # NIT buffering module
pseudo-device   clone

# connections for machine type 4 (SUN4_460 or Sun4_490)
controller      obmem 4 at nexus ?
controller      obio 4 at nexus ?
controller      vme16d16 4 at nexus ?
controller      vme24d16 4 at nexus ?
controller      vme32d16 4 at nexus ?
controller      vme16d32 4 at nexus ?
controller      vme24d32 4 at nexus ?
controller      vme32d32 4 at nexus ?
controller     ipi 4 at nexus ?        # IPI driver addresses


#
# The following (large) section describes which standard devices this
# kernel supports.
#

# Support for 4 IPI Panther IPI-2 Channel adaptors.
# This declares the channel driver part of the Panther support.

controller  isc0    at vme32d32 ? csr 0x01080000 priority 2
                vector isintr 0x4c
channel     is0 at isc0 ipi_addr 0x00000    # channel 0 slave 0
controller  isc1    at vme32d32 ? csr 0x01080400 priority 2
                vector isintr 0x4d
channel     is1 at isc1 ipi_addr 0x00100    # channel 0 slave 1
controller  isc2    at vme32d32 ? csr 0x01080800 priority 2
                vector isintr 0x4e
channel     is2 at isc2 ipi_addr 0x00200    # channel 0 slave 2
controller  isc3    at vme32d32 ? csr 0x01080c00 priority 2
                vector isintr 0x4f
channel     is3 at isc3 ipi_addr 0x00300    # channel 0 slave 3
```

(continued on next page)

(continued from previous page)

```
# Support for 4 IPI Panther Controllers with 8 drives each.
# This declares the disk driver part of the Panther support

# IPI Disk controllers and drives.

controller  idc0    at ipi ? csr 0x0000ff priority 2   # channel 0 slave 0
disk        id 0    at idc0 drive 0     # facility 0
disk        id 1    at idc0 drive 1
disk        id 2    at idc0 drive 2
disk        id 3    at idc0 drive 3
disk        id 4    at idc0 drive 4
disk        id 5    at idc0 drive 5
disk        id 6    at idc0 drive 6
disk        id 7    at idc0 drive 7

controller  idc1    at ipi ? csr 0x0001ff priority 2 # channel 0 slave 1
disk        id 0x10 at idc1 drive 0     # facility 0
disk        id 0x11 at idc1 drive 1
disk        id 0x12 at idc1 drive 2
disk        id 0x13 at idc1 drive 3
disk        id 0x14 at idc1 drive 4
disk        id 0x15 at idc1 drive 5
disk        id 0x16 at idc1 drive 6
disk        id 0x17 at idc1 drive 7

controller  idc2    at ipi ? csr 0x0002ff priority 2 # channel 0 slave 2
disk        id 0x20 at idc2 drive 0     # facility 0
disk        id 0x21 at idc2 drive 1
disk        id 0x22 at idc2 drive 2
disk        id 0x23 at idc2 drive 3
disk        id 0x24 at idc2 drive 4
disk        id 0x25 at idc2 drive 5
disk        id 0x26 at idc2 drive 6
disk        id 0x27 at idc2 drive 7

controller  idc3    at ipi ? csr 0x0003ff priority 2 # channel 0 slave 3
disk        id 0x30 at idc3 drive 0     # facility 0
disk        id 0x31 at idc3 drive 1
disk        id 0x32 at idc3 drive 2
disk        id 0x33 at idc3 drive 3
disk        id 0x34 at idc3 drive 4
disk        id 0x35 at idc3 drive 5
disk        id 0x36 at idc3 drive 6
disk        id 0x37 at idc3 drive 7
```

(continued on next page)

```
    (continued from previous page)

#
# Support for the SCSI-3 host adapter.  Same device support as above.
#
controller  si0 at vme24d16 ? csr 0x200000 priority 2 vector siintr 0x40
controller  si1 at vme24d16 ? csr 0x204000 priority 2 vector siintr 0x41
tape        st0 at si0 drive 32 flags 1
tape        st1 at si0 drive 40 flags 1
tape        st2 at si1 drive 32 flags 1
tape        st3 at si1 drive 40 flags 1
#disk       sf0 at si0 drive 49 flags 2
disk        sd0 at si0 drive 0 flags 0
disk        sd1 at si0 drive 1 flags 0
disk        sd2 at si0 drive 8 flags 0
disk        sd3 at si0 drive 9 flags 0
disk        sd4 at si0 drive 16 flags 0
disk        sd6 at si0 drive 24 flags 0
#
device      zs0 at obio ? csr 0xf1000000 flags 3 priority 3
#
device      zs1 at obio ? csr 0xf0000000 flags 0x103 priority 3
#
device      ie0 at obio ? csr 0xf6000000 priority 3
device      ie2 at vme24d32 ? csr 0x31ff02 priority 3 vector ieintr 0x76
#
# Support for monochrome memory frame buffers on various machines.
#
device      bwtwo0 at obio 4 csr 0xfb300000 priority 4

device      des0 at obio ? csr 0xfe000000
```

3.2. Customizing a Kernel

Two basic parameters may need to be tuned in a SPARCserver 490 kernel:

- number of ports (if you have ALM-2s)
- maxusers value (if you are supporting more than 24 clients)

Step 1: Adjust Number of Ports

In the kernel configuration file, the line

```
pseudo-device    mcpa64
```

needs to be adjusted to include all ALM-2 serial ports that are likely to be used (if you have no ALM-2s, you could delete or comment these lines out). For any kernel that is expected to support any of the upper four ALM-2 boards (mcp4 through mcp7), this line should be changed to

```
pseudo-device    mcpa128
```

to provide async protocol support for the whole set. Note that if you use a higher numbered ALM, then the "mcpa" number must be great enough to handle that ALM and all lower numbered boards, installed or not.

Create /dev Entries

When adding ALM-2s, you must create the /dev entries, as follows:

```
# cd /dev
# MAKEDEV mcp0 mcp1 mcp2 ...
```

where an mcp*n* entry is specified for each ALM-2, up to a max of mcp7. The `/dev` entries created by this command are the names which must be entered into `/etc/ttytab`.

Step 2: Increase maxusers

The line

```
    maxusers    24
```

must be increased to reflect the actual load on the system. The number of streams allocated is based on this number, so for proper allocation any gettys running on serial ports should be considered active sessions, and if any lines are running both dialin and dialout service via the upper 128 minor numbers, the dialout should be considered an additional user.

The calculation for maxusers is generally:

number of framebuffer sessions (i.e., windows and other tools, or one for a nonwindow login on the console),

plus

number of network sessions (`telnet`, `ftp`, `rsh`, and `rlogin` sessions to or from this host)

plus

number of serial ports with `gettys` running on them,

plus

maximum number of concurrent dialout (`tip` and `uucp`) sessions.

Then add a few, and round up to something nice and round, because you will probably underestimate, and the cost of the extra kernel size is not much when you have large memories (32 MB and up) and disks (600 MB and up). In general, systems with eight ALM-2 boards will also tend to have larger physical memories and larger, faster disks, so any error in maxusers should be on the high side.

Make Entries in /etc/ttytab

You will have to manually make entries in the `/etc/ttytab` file. See the *System & Network Administration* manual, Section 11.3, "Adding a Terminal to your System."

The procedure for adding tty ports;

1. Determine the total number of logins to be supported.
2. Apply the algorithm described above for computing maxusers.
3. Rebuild the kernel, if required.
4. Make entries in `/dev` for the new tty ports.

   Do this with `/dev/MAKEDEV`. Each argument to MAKEDEV represents one peripheral board, e.g., "MAKEDEV mcp0" means make all tty port entries (16) for the first ALM-2, "MAKEDEV mcp7" means make all tty port entries (16) for the 8th ALM-2 board.
5. Make entries in `/etc/ttytab` for each tty port. The format is shown in System & Network Admin, section 11.3.

   Sample `/etc/ttytab` entries:

```
ttyh0   "/usr/etc/getty std.9600"       unknown         on secure
ttyh1   "/usr/etc/getty std.9600"       unknown         on secure
ttyh2   "/usr/etc/getty std.9600"       unknown         on secure
ttyh3   "/usr/etc/getty std.9600"       unknown         on secure
ttyh4   "/usr/etc/getty std.9600"       unknown         on secure
ttyh5   "/usr/etc/getty std.9600"       unknown         on secure
ttyh6   "/usr/etc/getty std.9600"       unknown         on secure
ttyh7   "/usr/etc/getty std.9600"       unknown         on secure
ttyh8   "/usr/etc/getty std.9600"       unknown         on secure
ttyh9   "/usr/etc/getty std.9600"       unknown         on secure
ttyha   "/usr/etc/getty std.9600"       unknown         on secure
ttyhb   "/usr/etc/getty std.9600"       unknown         on secure
ttyhc   "/usr/etc/getty std.9600"       unknown         on secure
ttyhd   "/usr/etc/getty std.9600"       unknown         on secure
ttyhe   "/usr/etc/getty std.9600"       unknown         on secure
ttyhf   "/usr/etc/getty std.9600"       unknown         on secure
ttyi0   "/usr/etc/getty std.9600"       unknown         on secure
```

Possible Error Messages

If too many `/etc/ttytab` terminals are enabled, or too many remote logins occur, compared to the maxusers setting in the config file, then the message

```
stropen: out of streams
```

may be displayed on the console and in the `/usr/adm/messages` log. This is an indication that you should increase the maxusers value, and re-build the kernel.

After configuring, building, and booting the new kernel, fire up the system and run with it for a while. Any messages relating to exhaustion of streams resources are a red flag that you may need to refigure the correct value for maxusers.

Monitoring Performance

Monitor long term performance with "netstat -m," and reduce resource allocations if it is shown that peak allocations never get near the allocated maxima. You will want to do this re-allocation of resources if the performance of the machine is suffering due to lots of memory being hogged by the kernel — especially if the cost of reconfiguring the kernel is low (low impact on users).

Bear in mind that allocation of data buffer resources may start to fail when the high water mark reaches 80% of the configured maximum, as the system tries to reserve some resources for high priority messages. No matter how careful you are to watch the resources, you always need some extra room, so don't tune your system too tightly; tune it so that the observed maximum numbers are between 50% and 75% of the absolute limit values in your kernel configuration file.

3.3. Configuring a Kernel

See Section 7.8, "Configuring a Custom Kernel" in *Installing the SunOS 4.0.3* for instructions for configuring (building) a kernel after you have customized the kernel configuration file.

4

IPI8-1000 Disk Drive and ISP-80 Controller

IPI8-1000 Disk Drive and ISP-80 Controller 57

4.1. Description and Configuration Options 57
Definition of Terms 57
4.2. Addressing and Naming Conventions 57
Physical Addresses 57
Device Names 58
Boot Address Format 58
4.3. Default Boot from IPI8-1000 Disk 58
Using the q Command 58
Using the eeprom Command 59
4.4. Rebooting the System After Swapping a Disk Drive 61
4.5. IPI Error Messages 61
Basic Error Message Forms 61
Errors where the drive can be identified and there is a particular disk partition in use 61
For operations on the whole drive (such as format), where no partition information would be relevant 61
For operations to the controller, where the drive number isn't significant 62
<operation> field 62
Messages from the IPI Controller or IPI Drive 62
Message/Microcode Exception (from IPI Controller) 62
Intervention Required (from IPI Drive) 63
Alternate Port Exception (from IPI Drive) 63

Machine Exception (from IPI Drive/Controller) 63
Command Exception (from IPI Controller) 63
Conditional Success (from IPI Drive/Controller) 63
Incomplete (from IPI Controller) 63
Errors reported by the IPI Driver 63

SPARCserver 490
Software Installation Checklist

This checklist outlines the primary steps to follow in installing SPARCserver 490 (SunOS 4.0.3) software.

Step 1: Check contents of this box.

This box should contain the following:

- SPARCserver 490 (SunOS 4.0.3) installation tape
- *SPARCserver 490 Software Installation Checklist* (this document)
- *SPARCserver 490 (SunOS 4.0.3) Release Manual*
- *Read This First SPARCserver 490 (SunOS 4.0.3)*

If you ordered your SPARCserver 490 with the SunOS 4.0.3 tape and documentation option, the box will also contain:

- SunOS 4.0.3 Sun-4 installation tape set
- *Read This First SunOS 4.0.3*

Step 2: Read the *Read This First SPARCserver 490 (SunOS 4.0.3).*

Read this document carefully; it contains the latest information about the software, including descriptions of known bugs. It also provides pointers for a successful installation and tells you how to get help if you need it.

Step 3: Install software.

Follow the software installation procedures in Chapter 2 of the *SPARCserver 490 (SunOS 4.0.3) Release Manual.* The primary installation steps are:

1. Boot the miniroot from the SPARCserver 490 (SunOS 4.0.3) installation tape.
2. Run SunInstall to install SunOS 4.0.3 software.
3. Run `sunupgrade` to install SPARCserver 490 (SunOS 4.0.3) software.

Note that the *SPARCserver 490 (SunOS 4.0.3) Release Manual* lists all of the documentation included with your SPARCserver 490 order. Refer to the appropriate documents, as needed.

Sun Microsystems, Inc. • 2550 Garcia Avenue • Mountain View, CA 94043 • 415-960-1300

Part No: 800-4713-10
Revision A of 11 December 1989

4

IPI8-1000 Disk Drive and ISP-80 Controller

4.1. Description and Configuration Options

The IPI8-1000 disk drive is a high-capacity, 8" disk drive. The drive has an unformatted capacity of 1.2 gigabytes and formats to 1.03 gigabytes. Theoretically, you can have up to four high-performance, intelligent IPI controllers (ISP-80) on a SPARCserver 490, supporting up to eight one-gigabyte disk drives each, giving you a maximum of 32 disk drives, or 32 gigabytes of storage. Note that this maximum configuration may not be available at the present time; check with your Sun Sales Representative for the currently-available configurations.

Definition of Terms

- **IPI:** IPI stands for Intelligent Peripheral Interface; IPI defines a device-specific command set. IPI also specifies use of the IPI physical level, which defines the cables, connectors, and drivers/receivers to be used. The physical level specification also defines the low level bus protocol and state sequences on an IPI cable.
- **channel:** the interface between the channel adapter in the CPU board and an IPI *slave*. Note that there is no hardware channel adapter in the current configuration, but that the concept of a channel adapter has been implemented in the software (thus allowing for future growth).
- **slave:** a controller for IPI *facilities*; a slave is identified by its VME bus address.
- **facility**: an IPI device, such as a disk drive; a facility is identified by its address, which is set on the IPI8-1000 disk drive by depressing a button on the front panel of the drive.

4.2. Addressing and Naming Conventions

For historical and technical reasons, physical addresses, device names, and boot names are similar, but not exactly the same. Each of the conventions are described below.

Physical Addresses

The physical address of an IPI device is composed of three hexadecimal digits, in the format

<channel> <slave> <facility>

Theoretically, there can be one channel (0), four slaves per channel (0-3), and up to eight facilities per slave (0-7), for a total of 32 drive addresses. The range of physical addresses is id000 to id037.

Device Names

Device names, which are included on the installation tape, are of the format

`id` <channel> <slave> <facility> <disk partition>

or

`rid` <channel> <slave> <facility> <disk partition>

To accommodate future growth, the range of device names included on the installation tapes allows for one channel (0), four slaves (0-3), eight facilities per slave (0-7), and eight partitions per facility (a-h) for both character (`id`) and raw (`rid`) devices, for a total of 512 device names, in the ranges `/dev/id000a` to `/dev/id037h` and `/dev/rid000a to /dev/rid037h`.

Note that IPI device names **cannot** be abbreviated, so that the command

```
dkinfo id000a
```

cannot be shortened to

```
dkinfo id0a
```

Boot Address Format

When you copy the boot program from the installation tape to an IPI8-1000 disk and boot from disk, the format of the "boot address" is

`id`(<slave>,<facility>,<partition>)

with partitions designated 0 to 7 rather than a to h. You can only boot from slave/controller 0 at this time, with an eight-facility/disk maximum, so the range is id(0,0,0) to id(0,7,7) (from slave/controller 0, facility/drive 0, partition a, to slave/controller 0, facility/drive 7, partition h). Note that the channel is not included in the boot address format; since there is only one channel available, this is inconsequential.

4.3. Default Boot from IPI8-1000 Disk

If you have only IPI8-1000 disks on your system, booting by default will be from id000a. If you have other types of disks on your system and you want to boot from an IPI8-1000 disk, or if you have only IPI8-1000 disks and you want to boot from somewhere other than id000a, modify the EEPROM (Electronically-Erasable Programmable Read-Only Memory chip) as follows. Two methods are shown: using the `q` command from the boot PROM prompt (>); using the `eeprom` command while the system is up.

Using the q Command

Use this method right after installing SPARCserver 490 software, as described in Chapter 2, to be sure the system boots from the proper disk and partition.

1. Select the desired IPI disk and partition to boot from, e.g., `id002a` (facility/drive 2, partition a).
2. Convert this to boot address format, e.g., `id(0,2,0)`.

   Next you will write this boot address into the appropriate fields of the

EEPROM. The fields to be modified are the device name (`id`) and the three numeric values (controller, drive, and partition numbers).

3. Modify the EEPROM.

   Use the `q` command to modify the EEPROM. Note that preceding the device-type characters by the at sign (@) converts the character from ASCII to hexadecimal format.

```
>q 18                    (boot device address start)
 ....? 12 <Return>       (tells monitor that default boot device follows)
 ....? @i <Return>       (1st character of device type, in hex)
 ....? @d <Return>       (2nd character of device type, in hex)
 ....? 0 <Return>        (controller number)
 ....? 2 <Return>        (drive number)
 ....? 0 <Return>        (partition number)
 ....? <space> <Return>          (end of change)
```

4. Boot the system:

```
> b
```

Using the eeprom Command

If you decide to change the default boot device after bringing the system up, do the following:

1. Select the desired IPI disk and partition to boot from, e.g., `id002a`.
2. Convert this to boot address format: `id(0,2,0)` (slave 0, facility 2, partition a, with channel 0 understood).
3. Check current EEPROM settings.

   Type `eeprom` to see the current settings of the EEPROM; note in particular the settings for `default_boot` and `bootdev`; these are the two settings which need to be changed.

```
tyler% eeprom <Return>
hwupdate=Wed Dec 31 16:00:00 1969
memsize=4
memtest=4
scrsize=1152x900
watchdog_reboot=false
default_boot=false
bootdev=(0,0,0)
 . . .
 . . .
```

4. Become superuser.
5. Set `default_boot` to "true":

```
#eeprom default_boot=true <Return>
```

This sets the EEPROM so that the default boot device comes from the EEPROM — specifically from the `bootdev` field.

6. Specify default boot device and partition:

```
#eeprom bootdev="id(0,2,0)"<Return>
```

7. Confirm proper settings.

   Confirm that `default_boot` is set to "true" and that the correct default boot device is specified:

```
#eeprom <Return>
hwupdate=Wed Dec 31 16:00:00 1969
 . . .
 . . .
default_boot=true
bootdev=id(0,2,0)
 . . .
 . . .
```

4.4. Rebooting the System After Swapping a Disk Drive

When an IPI disk drive is newly attached to the ISP-80 Controller, or if any IPI drive address is changed, (by way of the front panel controls), the system must be rebooted before the drive is accessed by software; otherwise, there is a danger that SunOS will have cached information which was applicable to a different drive which had the same address. This cached information will be wrong for the new or newly-addressed drive, thus causing possible data loss.

To prevent such loss, perform the following procedure:

1. Halt the system (with `/etc/halt`, for instance).
2. Attach drive cables and connectors, if necessary.
3. Change IPI drive addresses (via front panel), if desired.
4. Power off and power on all drives which had address changes, or power on any new drives.
5. Boot the system.

4.5. IPI Error Messages

This writeup describes the types of messages (usually error messages) reported by the IPI disk driver (id). For a full description of any particular message, see the id(4S) man page.

Basic Error Message Forms

IPI devices are addressed with the name syntax `/dev/`*idcsfp*, where:

id is the driver name for IPI disks.
c is the channel number (always 0 for ISP-80 controllers)
s is the slave number. This is sometimes called the controller number. For channel 0, the possible slave numbers are 0, 1, 2, and 3.
f is the facility number. This is sometimes called the drive number. The possible facility numbers are 0 to 7. This number matches the IPI drive address as selected on the drive front panel.
p is the traditional SunOS partition, a-h.

The device name is never abbreviated; the first drive on the first ISP-80 is id000. All block numbers are in units of 512-byte sectors.

Errors where the drive can be identified and there is a particular disk partition in use

```
id001b: block 123 (100123 abs): <operation>:  <message>
```

The "block nnn" value is the block number within the partition. The "nnn abs" value is the block number from the beginning of the disk.

For operations on the whole drive (such as format), where no partition information would be relevant

```
id001: block 123: <operation>: <message>
```

The "block nnn" value is the block number from the beginning of the disk.

For operations to the controller, where the drive number isn't significant

```
idc0: <operation>: <message>
```

<operation> field

For all the above message syntaxes, the <operation> field will be the IPI operation being performed: no-op, Attributes, Report Status, read, write, format, read defect list, write defect list, reallocate, allocate restore, or async interrupt. It can also have the value "op <opcode>," if for some reason the operation isn't in an internal table of operation names. In this case the <opcode> field contains the hexadecimal IPI-3 opcode.

Messages from the IPI Controller or IPI Drive

IPI messages can be caused by:

- an error responses from the IPI Controller
- an error detected within the IPI Driver

IPI messages follow the IPI ANSI specification. These messages are not classified by severity, however. A harmless condition can have a message which looks very similar to a fatal condition.

The syntax of <message> can be <type> <detail> or it can be just <detail>. The values of <type> are shown following. For the full text of <detail> messages, refer to the man pages id(4S), ipi(4S), and is(4S).

IPI messages caused by error responses from the ISP-80 controller are broken down into the following classifications, described below:

Response Type	Source
Message/Microcode Exception	Controller
Intervention Required	Drive
Alternate Port Exception	Drive
Machine Exception	Drive
Command Exception	Controller
Conditional Success	Drive
Incomplete	Controller

These responses are described below.

Message/Microcode Exception (from IPI Controller)

This is used by the controller to print messages on the SunOS console. The messages can indicate anything from the controller firmware revision level to a panic by the firmware.

The driver will sometimes print these messages with the string "ctlr message" prefixed.

Intervention Required (from IPI Drive)

This indicates a condition such as not-ready or busy at the drive. This could be caused by a loose cable or by the drive being manually stopped or powered off. Hardware problems in the drive, and especially overheating, have been known to cause these.

User intervention is probably required to recover from this type of problem.

Alternate Port Exception (from IPI Drive)

This indicates the drive is reserved or busy with an operation on the other IPI-2 port. This shouldn't happen with ISP-80 under correct configurations. Certain drive failures may also generate this error.

Machine Exception (from IPI Drive/Controller)

This classification is used to report serious errors and simple status changes. It includes conditions such as a drive becoming ready or no-longer busy (usually by an asynchronous interrupt). It includes ECC Uncorrectable Data Checks as well.

The IPI Driver retries some of these IPI Drive errors. If the status includes "Operation Timeout," "Physical Interface Check," or "Data Overrun," the IPI Driver retries the command, and an error is not returned to the calling program unless retry fails. A subsequent message will be issued if retry fails.

Command Exception (from IPI Controller)

This message is displayed when the controller finds something wrong with a command issued by the driver. The message will say what was wrong, such as "Invalid Packet Length" or "Invalid Opcode" or "Missing Parameter." This error probably indicates a broken IPI controller, or (occasionally) another broken hardware component. Certain of these errors may also be generated by entering incorrect information about a disk information utility program such as `format`.

Note that the controller retries most other machine exception errors. Blocks on which "Uncorrectable Data Check " errors persist may be fixed by performing a repair operation on the block; data from such blocks, however, will not be recovered.

Conditional Success (from IPI Drive/Controller)

This indicates a condition in the drive that may need attention, but the command completed successfully. It should be considered a warning. For example, the condition may be "Soft Error," "Data Correction Performed," or "Data Retry Performed," which are the only conditions for which repair operations will be effective. If the condition persists on a particular block, perhaps that block should be reallocated (via format's `repair` command).

Incomplete (from IPI Controller)

Some portion of a command finished, but it is not entirely complete. Never issued by the current IPI Controller.

Errors reported by the IPI Driver

The following errors are reported by the id driver, and aren't directly due to responses from the IPI Controller or slave. This list is not comprehensive; it lists some of the more common messages. For a complete list, see the id(4S) man page.

```
idc0: ctlr message: 'Warning:  bad EEPROM checksum '
```

This indicates that the controller's EEPROM has failed, or more likely, was not initialized correctly in manufacturing.

```
idc0: ctlr message: 'FW revision date = 7/5/89 , level = 4 '
```

This simply gives the revision level of the firmware (the revision date and level may be different than that shown above).

```
idc0: ctlr message: 'Warning:  drive 4 may require formatting '
```

This indicates an unformatted drive, or an obsolete and unsupported format on the drive.

```
id004: warning: label doesn't match IPI attribute info

id004: label ncyl 1630 pcyl 1634 acyl 002 head 015 sect 082 rpm 3600

id004: geom  ncyl 1632 pcyl 1633 acyl 001 head 015 sect 082 rpm 3600
```

These three messages indicate a drive which is labeled with a non-standard label. The second line gives the geometry information from the label, the third line gives the geometry information from the IPI attributes. The information from the label is used and the drive may seem to function normally, but it should be reformatted and relabeled correctly.

```
id004a: block 12345 (12345 abs): read: missing interrupt - attempting recovery
```

An operation took much longer than expected. The driver will test to see if the controller is still responding to commands. If it is, it will stop sending new commands to it for a while and give it a chance to finish the outstanding commands. If it doesn't finish, or was unresponsive, the controller will be reset and all commands will be retried. Messages will indicate whether this recovery succeeded or not.

```
id004a: block 12345 (12345 abs): read: missing interrupt - recovery in progress
```

An operation took much longer than expected, but recovery was already underway because of a prior missing interrupt.

```
idc0:   Recovery complete.
```

Missing interrupt recovery has completed successfully.

```
idc0:   Recovery failed.  Controller marked unusable.
```

Missing interrupt recovery wasn't able to reset and retry all operations; the controller and/or drives have a hard failure of some kind.

5

SCSI Devices

SCSI Devices **69**

5.1. SCSI Tape Drive Configuration Options 69
5.2. Front-Load 1/2″ Tape Drive 69
5.3. 2.3-Gbyte 8mm Tape Drive 69
5.4. 150-Mbyte 1/4″ Tape Drive 69
5.5. Determining the Tape Drive Device Name 69
Tape Drive Configurations 70
Using the mt Command 70
5.6. Data Format Addressing 71
150-Mbyte 1/4″ Tape Drive 71
2.3-Gbyte 8mm Tape Drive 71
Front-Load 1/2″ Tape Drive 71
5.7. Boot Address Formats 72

5

SCSI Devices

5.1. SCSI Tape Drive Configuration Options

The SPARCserver 390 supports three kinds of SCSI tape drives, which can be configured in any combination of one, two, three, or four drives. Note that this maximum configuration may not be available at the present time; check with your Sun Sales Representative for the currently-available configurations.

Note also that installation software is distributed only on 1/4" and 1/2" tapes, so you will have to have a 1/4" or 1/2" drive available on your system (unless you plan on doing a remote installation, which is explained in Chapter 2).

- 150-Mbyte 1/4-inch Tape Drive
- Front-Load 1/2" Tape Drive (FLT)
- 2.3-Gbyte 8mm Tape Drive

5.2. Front-Load 1/2″ Tape Drive

This is a newly-supported device, which accepts standard 1/2" tape reels, of various sizes (5 inches, 7 inches, and 9 inches), of various capacities, depending on the size of the reel and the recording density. The Front-Load Tape is sometimes called the "FLT."

5.3. 2.3-Gbyte 8mm Tape Drive

This is a newly-supported device, with a high storage capacity (2.3 gigabytes) and a small cartridge (2.5 inches by 4.5 inches by 0.5 inches). The format of the cartridge is 8mm, which is different than the 1/4-inch cartridges Sun has previously shipped.

5.4. 150-Mbyte 1/4″ Tape Drive

This is the 150 Mbyte 1/4-inch cartridge drive that is found in many existing Sun products. The 1/4-inch drive has a capacity of 150 megabytes.

Note: the 150 Mbyte drive can read standard 60 Mbyte cartridges, but it cannot write to them. The only writable cartridge as of the publication date of this document is the 3M DC6150.

5.5. Determining the Tape Drive Device Name

During installation of the SunOS software and when you use `tar` and other commands, you will need to know the "device name" — `st0, st1, st2, st3` — of the tape drive in which the tape is loaded.

If you have just one tape drive — a 1/4-inch cartridge drive, a front-load tape drive, or an 8mm tape drive — the device name is always going to be st0. This

will select the default data format; see "Data Format Addressing" below for other data format specifiers.

Tape Drive Configurations

The table below shows possible SCSI tape drive configurations in the Primary Cabinet and Expansion Cabinet. The types of drives are:

- *1/4"* is the 150-Mbyte 1/4-inch Tape Drive
- *FLT* is the Front-Load 1/2-inch Tape Drive
- *8mm* is the 2.3-Gbyte 8mm Tape Drive

For each combination of drives, the device name of each drive is indicated by `st0, st1, st2,` or `st3`. Note that "SCSI Left" and "SCSI Right" refer to the left side and right side as you look at the front of the cabinet.

Primary Cabinet			*Expansion Cabinet*	
SCSI Left	*SCSI Right*	*Upper FLT*	*Upper FLT*	*Lower FLT*
1/4"(st0)				
1/4"(st0)		FLT(st1)		
1/4"(st0)		FLT(st1)	FLT(st2)	
1/4"(st0)		FLT(st1)	FLT(st2)	FLT(st3)
1/4"(st0)			FLT(st2)	
1/4"(st0)			FLT(st2)	FLT(st3)
8mm(st0)				
8mm(st0)	1/4"(st1)			
8mm(st0)	1/4"(st1)	FLT(st2)		
8mm(st0)	1/4"(st1)		FLT(st2)	
8mm(st0)	1/4"(st1)		FLT(st2)	FLT(st3)
8mm(st0)	8mm(st1)	FLT(st2)		
8mm(st0)	8mm(st1)		FLT(st2)	FLT(st3)
8mm(st0)		FLT(st1)		
8mm(st0)			FLT(st2)	
8mm(st0)			FLT(st2)	FLT(st3)
8mm(st0)		FLT(st1)	FLT(st2)	FLT(st3)
		FLT(st0)		
		FLT(st0)	FLT(st2)	
		FLT(st0)	FLT(st2)	FLT(st3)
			FLT(st2)	
			FLT(st2)	FLT(st3)

Using the mt Command

To be sure of the device name of a tape drive, use the `mt` command, which will tell you what type of drive corresponds to each device name. Note that you have to have a tape in the drive for this command to work.

1. Insert a tape into each drive to be tested.

2. Execute `mt` command, substituting 0, 1, 2, or 3 for the question mark (?):

```
% mt -f /dev/rst? status
```

This returns the tape drive manufacturer and model number or the drive format, which should tell you which drive is which. The 150 Mbyte 1/4-inch drive will give you a response like this, for instance:

```
Archive QIC-150 tape drive:
   sense key(0x0)= no sense   residual= 0   retries= 0
   file no= 0   block no= 0
```

5.6. Data Format Addressing

The three SCSI tape devices read and write in various formats; the format is sometimes determined by the device name, as shown below for the Front-Load Tape Drive.

150-Mbyte 1/4″ Tape Drive

The 150-Mbyte 1/4-inch tape drive *writes* in 150 Mbyte format (QIC-150) — or 120 Mbyte (QIC-120), with certain types of tape. The drive auto-selects on *reads*, meaning that it can read 60 Mbyte (QIC-11 or QIC-24), 120 Mbyte (QIC-120), or 150 Mbyte (QIC-150)tapes.

2.3-Gbyte 8mm Tape Drive

The 2.3-Gbyte 8mm tape drive has only one format, for both reads and writes.

Front-Load 1/2″ Tape Drive

The front-load tape drive reads and writes in 800bpi, 1600bpi, or 6250bpi format, depending on the device name:

Where on Bus	***Format***		
	800bpi (NRZI)	***1600bpi (PE)***	***6250bpi (GCR)***
1st on Bus (primary cabinet)	st0	st8	st16
2nd on Bus (primary cabinet)	st1	st9	st17
1st on Bus (expansion cabinet)	st2	st10	st18
2nd on Bus (expansion cabinet)	st3	st11	st19

5.7. Boot Address Formats

In the PROM Monitor, SCSI device names are not written the same as SunOS `/dev` names. Following is a table showing the mapping from SunOS `/dev` names to PROM Monitor names. The PROM Monitor names are used when booting.

Tape Device	PROM Monitor Name
/dev/st0	st(0, 20, x)
/dev/st1	st(0, 28, x)
/dev/st2	st(1, 20, x)
/dev/st3	st(1, 28, x)
where "x" is the number of the file from beginning of the tape.	
Disk Device	**PROM Monitor Name**
/dev/sd0	sd(0, 0, y)
/dev/sd1	sd(0, 1, y)
/dev/sd2	sd(0, 8, y)
/dev/sd3	sd(0, 9, y)
/dev/sd4	sd(0,10, y)
/dev/sd6	sd(0,18, y)
where "y" is a number from 0 to 7 to identify the disk partitions a to h.	

The Boot tape name syntax is **st**(*ha, sa, file*) where:

ha means which SCSI Host Adaptor,
sa is a hex number which is the SCSI bus address of the tape device,
file is a hex number of the tape file from beginning of the tape.

The Boot disk name syntax is **sd**(*ha, sa, partition*) where:

ha means which SCSI Host Adaptor,
sa is a hex number which is the SCSI bus address of the disk device,
partition is a hex number from 0 to 7, which is the disk partition.

VAM UNIT NUMBER

6

SunNet Ethernet/VME Controller

SunNet Ethernet/VME Controller **75**

6.1. Description and Configuration 75
6.2. Configuring an Ethernet Gateway 75

6

SunNet Ethernet/VME Controller

6.1. Description and Configuration

The SPARCserver 490 has a built-in Ethernet port, `ie0`, and an optional second port, `ie2`, which is housed in the SunNet Ethernet/VME Controller. The second port allows you to add a second Ethernet connection, and to use the SPARCserver 490 as a router between two networks.

Unlike most hardware devices, network interfaces do not have entries in the `/dev` directory; they have three-character names — two letters followed by a digit. The state and activity of a network interface port can be checked with the commands `/etc/ifconfig` and `/usr/ucb/netstat`.

```
tyler% ifconfig ie0
ie0: flags=63<UP,BROADCAST,NOTRAILERS,RUNNING>
        inet 129.144.50.5 netmask ffffff00 broadcast 129.144.50.0

tyler% netstat -n -I ie0
Name  Mtu   Net/Dest    Address       Ipkts   Ierrs Opkts   Oerrs Collis Queue
ie0   1500  129.144.50  129.144.50.5 200154  8     33833   0     14010
```

6.2. Configuring an Ethernet Gateway

See Section 2.3 in this manual for instructions for configuring an Ethernet gateway.

A complete description of the procedure for setting up a router to join two networks is in Chapter 12, "The SunOS Network Environment" in *System and Network Administration Manual*. Also, see the networks(5) man page.

7

Diagnostic Testing

Diagnostic Testing **79**

7.1. Enhanced Diagnostic Software 79
The sundiag Program 79

7

Diagnostic Testing

7.1. Enhanced Diagnostic Software

SunOS 4.0.3 introduced a new diagnostic system, `sundiag`, which is a SunView-based user interface that tests system devices and peripherals, including the new devices supported on the SPARCserver 490:

- ISP-80 Controller and IPI8-1000 Disk Drives
- Front-Load 1/2-inch Tape Drive
- SunNet Ethernet/VME Controller

The sundiag Program

The `sundiag` program is an online system exerciser for testing peripheral devices. Incorporating a SunView-based interface, `sundiag` can be used on any Sun-2, Sun-3, or Sun-4 hardware configuration running SunOS 4.0 and later. Configure the system kernel to support all peripheral devices to be tested.

`sundiag` executes system tests that formerly were performed by `sysdiag`, and runs all `sundiag` tests written for new Sun products. Read the *Sundiag User's Guide*, Part Number 800-3817, for complete information about using this test program. The `sundiag` program is in the `/usr/diag/sundiag` directory.

Note that Sundiag is loaded on your system only if you choose to install the optional "User_Diag" category from the SunOS 4.0.3 and SPARCserver 490 tapes as part of the installation process.

This information supplements or replaces that found in the *Sundiag User's Guide*, PN 800-3817.

NOTE *If you have a tape drive in your system, load a blank tape before you start Sundiag. If you fail to do so, Sundiag will show*

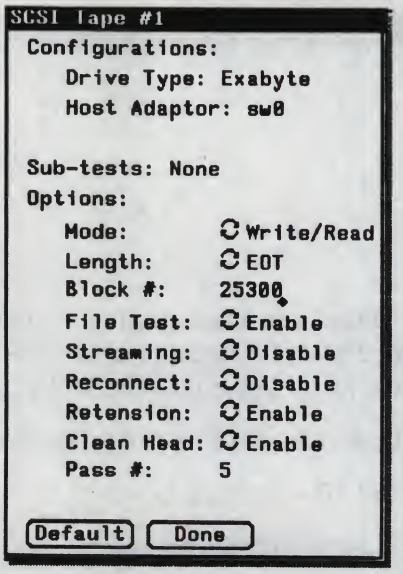
```
drive type:unknown
```

on the option menu for the tape test.

Sundiag now supports 2.3-Gbyte 8mm tape drive (Exabyte) and front-load 1/2-inch tape drive (HP-88780) testing. The option pop-ups for those drives are shown in the examples below.

On pages 45 and 46, under the *tapetest* description, a tape cleaning warning is discussed. This feature has been replaced with an option pop-up choice. Please ignore the text that says a head cleaning warning will pop up at 8-hour intervals.

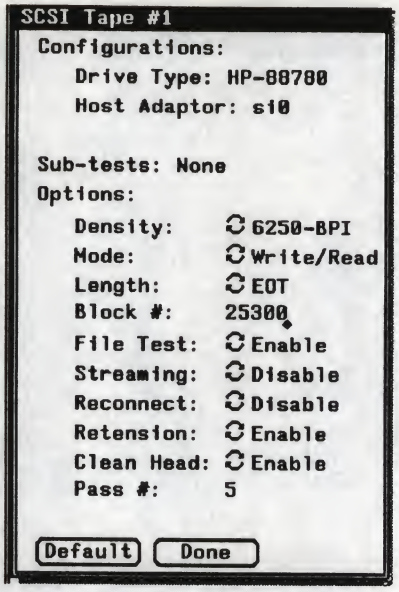
Instead, when you click on the tape test **Option** button, a menu like the following is displayed:



If you want to enable head cleaning, click on the circular arrows after **Clean Head** to **enable**. Then, after **Pass #** enter the number of test passes Sundiag should execute before suspending testing to provide time to clean the tape drive head.

The example above depicts the option menu for a 2.3-Gbyte 8mm tape drive. This menu differs from other tape drive option menus in that it has no format or reconnect option choices. The **Clean Head** and **Retension** options are new with this version of Sundiag. When you cycle to enable the retension option, the drive retensions the tape. The head cleaning option was described above.

If the drive is a front-load tape drive (FLT), the following type of option menu will be displayed:



Besides the **Clean Head** option, you will see a **Retension** option. *Make sure that the retension option reads* **Disable** when testing the front-load tape drive. This option is not supported and will be removed from future releases.

A

PROM User's Manual Addenda and Errata (2)

PROM User's Manual Addenda and Errata (2) **85**

A.1. New PROM Monitor Features 85
Entering Decimal Values 85
Entering ASCII Characters 85
New `^i` Feature 85
A.2. New Boot-Up Features 86
The Power-Up Display 86
Revising the Banner 86
Automatic IPI Boot-Up 88
Sun-3/80 Diagnostic Boot-Up 88
PROM Monitor Security Feature 89
The Password 89
Changing Security Modes 91
EEPROM Layout for PROM Security 91

A

PROM User's Manual Addenda and Errata (2)

This text assumes that you have read the *PROM User's Manual Addenda and Errata* that was shipped with SunOS Release 4.0.3. This information replaces the first five pages of that document and also acts as addenda/errata to the 4.0.3 *Read This First* and the 4.0.3 *Release Manual*.

The *PROM User's Manual* describes PROM monitor commands, self-tests and extended tests for Sun-2 through Sun-386i systems. This text also adds to or modifies the information found there. This addenda applies to the SPARCserver 490 and Boot PROM revisions 2.8 through 3.0.

A.1. New PROM Monitor Features

In the past, all numerical PROM monitor commands were entered in hexadecimal. If you have PROM version 2.8, or a Sun-3/400 series system, a Sun-3/80 or SPARCsystem 330, you may now enter decimal or ASCII values after the PROM monitor prompt (>). This feature is particularly useful when using the monitor `q` command to program the EEPROM, which sometimes requires that you convert letters and decimal numbers to hexadecimal values before you enter them.

Entering Decimal Values

To enter a decimal value after a PROM monitor command, simply precede the value with the "%" character:

```
>q 050 %20
```

Entering ASCII Characters

To enter an ASCII character, simply precede it with the "@" character:

```
>q 022 @i
>q 023 @e
```

If the value you enter is not preceded by a % or @ character, the monitor program treats that the value as hexadecimal.

New `^i` Feature

Entering the caret (`^`) and then an `i` when in PROM monitor mode (explained in the *PROM User's Manual*) now displays a Boot PROM copyright message in addition to the compilation information for the system firmware:

```
Copyright (c) 1985 - 1989 Sun Microsystems, Inc.
All Rights Reserved - World Wide
Compiled on 4/24/89 using hostname:/directory_path
```

A.2. New Boot-Up Features

This section describes the type of display that appears on the console screen when you power-up, and the EEPROM entries required to change the first line of that display, or the presence/appearance of the Sun logo.

An EEPROM option that ensures a boot from the Intelligent Peripheral Interface board is also described.

A special EEPROM entry needed in order to perform a diagnostic boot on a Sun-3/80 is described.

Finally, the new PROM monitor security feature is discussed.

The Power-Up Display

When you power-up your Sun system, a banner tells you what model you have, what Boot PROM revision is on your CPU board, how much main memory is installed, what your system serial number is, and what your Ethernet address is. Now, in addition to that information, your Host ID number is also displayed. Please note that, for SPARCsystems, your banner may say Sun SPARCstation rather than Sun Workstation. Here is an example of the new power-up display:

```
Sun Workstation, Model Sun-3/400 Series.
ROM Rev 3.0 12MB memory installed, Serial #xxx.
Ethernet address 1:2:30:4:55:66, Host ID xxxxxxxx.
```

Revising the Banner

Note that it is possible to change the Sun logo so that it appears three-dimensional, when a CG6 board is installed in a system with a Revision 3.0 Boot PROM. In addition, you may delete the first line of the banner shown above, and replace it with a banner and logo of your choice:

```
          SomeOtherCompany name
custom    ROM Rev 3.0 12MB memory installed, Serial #xxx.
 logo     Ethernet address 1:2:30:4:55:66, Host ID xxxxxxxx.
```

In order to change the first line of the Sun power-up banner or replace the Sun logo, you must change EEPROM parameters. Chapter 14 of the *PROM User's Manual* describes use of the PROM monitor program `q` command and provides the values and EEPROM offset addresses that must be changed in order to accomplish this power-up display change.

Some parameters may also be changed with the SunOS `eeprom` command.

You may use the SunOS `eeprom` command to specify a custom banner. To do so, become super-user and enter:

```
%su
enter super-user password
#eeprom custom_logo=true   (actually means custom_banner)
#eeprom banner="my new banner string"   (Don't forget the double quotes!)
#exit
%
```

Now, use a command such as `/etc/halt` and then power-cycle your system, and your new banner should display when you power-up. Note that the `eeprom` command does not provide for changing the bit-mapped logo.

NOTE *The EEPROM addresses and values entered with the PROM monitor* `q` *command are hexadecimal values.*

If you you use the PROM monitor **q** command to change the first line of the power-up display, you must enter the hexadecimal value **12** in the EEPROM location 020. Then you must enter replacement text in locations 068 through 0B7. You may use the new **@** symbol before each character, instead of converting them to hexadecimal as described in the *PROM User's Manual.*

If your system has a CG6 board installed, and you have a revision 3.0 Boot PROM, you may enter the value **0** in location 020, and and the value **6** in location 18F, and a special banner with a three-dimensional Sun logo will be displayed upon power-up.

If you want to replace the Sun logo with another bit-mapped display, you must enter a **12** in EEPROM location 18F. The replacement 64x8 bit-mapped display must be placed in EEPROM locations 290 through 48F.

Note that a hexadecimal **12** in location 020 causes the boot program to look for text in locations 068 through 0B7. If zeroes are in those locations and you have specified a custom banner with a 12(hex) in location 020, the first line of the power-up display will disappear the next time you power-up.

In addition, as indicated in the table that follows, if you enter a 12 in location 020 and then enter 0 or 6 in location 18F, the Sun logo will disappear and NO LOGO will be displayed on power-up.

If you enter a 12 in both locations 020 and 18F, you will be expected to place a string of custom banner text in locations 068 through 0B7, and a bit-mapped graphic in locations 290 through 48F. If zeroes are in those locations, the next time you power up, NO LOGO will appear and the first line of the power-up banner will disappear.

The EEPROM locations shown in the table below are valid **only for systems with a Revision 3.0 Boot PROM.**

To summarize the EEPROM locations that affect the power-up display:

Banner Location 020 Value	*Logo Location 18F Value*	*Logo Type*	*Power-Up Banner Type*
0	0	b/w Sun	Sun
0	6	Color 3D Sun For CG6 Only	Sun
0	12	Custom	Sun
12	0	No Logo	Custom
12	6	No Logo	Custom
12	12	Custom	Custom

NOTE *In location 020, any value other than* **12** *is treated as a zero.*

In location 18F, any value other than a **6** *or* **12** *is treated as a zero, and the 6 only affects systems that have CG6 boards installed. For all other systems, a 6 in location 18F is treated as a zero.*

Automatic IPI Boot-Up

If you want to ensure that your system *always* boots from the Intelligent Peripheral Interface board, you may use the **q** command to change EEPROM location 0x018 to the value **12**, and then locations 0x19-1D to contain the boot sequence

```
id(0,0,0)
```

The SunOS `eeprom` command provides a simple, interactive way of making the same change. To do so, become super-user, then enter

```
#eeprom
```

You will now view some of the configuration information present in the EEPROM. To set the boot device, you must enter:

```
#eeprom default_boot=true      (this means you want to specify the boot device)
#eeprom bootdev="id(0,0,0)"    (boot only from IPI disk 0, controller 0, partition 0)
```

To verify the change, enter the `eeprom` command again. You should now see these entries:

```
default_boot=true
bootdev=id(0,0,0)
```

Sun-3/80 Diagnostic Boot-Up

If you have a Sun-3/80, an EEPROM location determines whether you will power-up in normal or diagnostic mode. There is no diagnostic switch on the Sun-3/80. If you want to attach a terminal to serial port A and view the diagnostic self-test messages, you must first use the PROM monitor **q** command to enter **12** in EEPROM location 70B (hexadecimal).

If you want a normal boot, enter the value **6** in location 70B. All values other than 6 cause the Sun-3/80 to boot in diagnostic mode. (Use 0 instead of 6 if your

boot PROM is less than rev 3.)

PROM Monitor Security Feature

Chapter 10 of the *PROM User's Manual* describes the Sun-3 and Sun-4 EEPROM. There is now a Security Mode Select Feature, located at EEPROM address 0x492. This feature provides a *non-secure* mode that permits the use of all PROM monitor commands.

It also provides a *command secure* mode that permits the use of PROM monitor commands (other than **c**, for continue, or **b**, for boot, without parameters) only when a password is entered. In the command secure mode, you may operate your workstation normally, including powering down, booting, terminating with the L1-a command, and re-booting. You may not, however, perform any unusual operations, such as booting a non-standard kernel, running diagnostics, or changing EEPROM or CPU board memory contents, without entering a password.

Finally, this feature provides a *fully secure* mode that does not permit use of the PROM monitor (other than the **c** command with no parameters) without entering a password. To power-up, re-boot, or perform any other PROM monitor operation, you must supply a password. This mode allows you to control access to the workstation by turning it off. The workstation does not automatically boot on power-up.

CAUTION **In fully secure mode, even a default boot cannot be completed unless the password is entered. Once the SunOS is halted, you cannot restore it until you enter the correct password after the prompt. If the password is unknown, the system CPU board must be serviced as a failed board.**

The Password

If you should attempt to enter a PROM monitor command such as q, for example, on a system that is set for one of the secure modes, your interaction might look like this:

```
>q 020                    (you want to look at EEPROM offset address 020)
>Mon Pass:                (you enter the correct password)
>EEPROM 020: 00 ?         (the contents of location 020 are shown)
```

Or, if you enter an incorrect password, your interaction might look like this:

```
>q 020
>Mon Pass: (you enter the wrong password)
>Invalid
                (there is a slight delay)
>
```

You may now try again or enter an unprotected command.

To install or change a password, the system must be in non-secure mode, or you must know the existing password for a secure system. You then use the PROM monitor **q** command to enter the password in EEPROM offset location 490

NOTE *When you attempt to change the values stored in the EEPROM monitor password locations, you will be prompted with this message:*

```
Modifiying security location(s). Are you sure?(y/N)
```

If you enter **y** *for yes, the change you entered is written to the location shown. If you enter* **n** *for no, nothing is written to EEPROM, and the contents of the next location are displayed.*

To enter a monitor password, use the `@` character, described at the beginning of this document, in order to enter the letters that make up the password. If you do not use the `@` character, you must enter the hexadecimal equivalent of the letters. Following is an example of the way you would enter a password called *mypasswd* on a Sun-3 or SPARC -based system. You enter one letter per location, followed with : Return. To exit the command, you may enter any non-hexadecimal character, such as period, as shown.

```
>q 493
> EEPROM 493: 00? @m
Modifiying security location(s). Are you sure?(y/N) Yes
> EEPROM 494: 00? @y
Modifiying security location(s). Are you sure?(y/N) Yes
> EEPROM 495: 00? @p
Modifiying security location(s). Are you sure?(y/N) Yes
> EEPROM 496: 00? @a
Modifiying security location(s). Are you sure?(y/N) Yes
> EEPROM 497: 00? @s
Modifiying security location(s). Are you sure?(y/N) Yes
> EEPROM 498: 00? @s
Modifiying security location(s). Are you sure?(y/N) Yes
> EEPROM 499: 00? @w
Modifiying security location(s). Are you sure?(y/N) Yes
> EEPROM 49a: 00? @d
Modifiying security location(s). Are you sure?(y/N) Yes
> EEPROM 49b: 00? .
>
```

NOTE *If a password was already stored in locations 493-49a, hexadecimal values would appear in place of the zeroes in the example above.*

The password you enter must fill the eight bytes (locations 493-49a) with a character or a zero.

Note that changes to the security mode and password do not take effect until the PROM monitor mode is exited and then re-entered.

It is recommended that the password be changed before the security mode is changed. For more information on using the EEPROM **q** command, refer to the *PROM User's Manual*, Sun PN 800-1736, or the `Monitor(8S)` section of the *SunOS Reference Manual.*

Changing Security Modes

The EEPROM offset location 492 contains a value that determines the security mode. The table below shows the interpretation of values found in that location. The "0x" denotes a hexadecimal value.

0x1	command secure
0x5e	fully secure
all other values	non-secure

It is also recommended that the `chmod` and `/etc/chown` commands be used so that the `/dev/eeprom` device file may be accessed by the super-user. To accomplish this, enter:

```
%su
Password: enter your super-user password
# cd /dev
# /etc/chown root eeprom
# chmod 600 eeprom
```

EEPROM Layout for PROM Security

The following table shows the EEPROM offset locations for Sun-3 and SPARC -based systems.

Sun-3, SPARC Offset	*Field*	*Function*
0x490-1	bad_login	The bad login counter stores the number of invalid password attempts. The maximum value is 65535 and the counter does not roll over to 0.
0x492	secure	The values 1 and 0x5e correspond to command secure and fully secure, respectively. Any other value is non-secure.
0x493-a	password	If the password is shorter than 8 bytes, the password string is fenced with a null character.

B

Man Pages for SPARCserver 490 (SunOS 4.0.3) Release

Man Pages for SPARCserver 490 (SunOS 4.0.3) Release 95

B

Man Pages for SPARCserver 490 (SunOS 4.0.3) Release

Introduction

The following "man" pages are updated or added for the SPARCserver 490 (SunOS 4.0.3) release. Insert the attached pages into the *SunOS Reference Manual* (Part No. 800-1751). View these pages on-line with the `man` command — after installing SPARCserver 490 software, including the software category "man."

Section 4

`Intro`(4) — updated for this release
`dkio`(4S) — updated for this release
`id`(4S) — new for this release
`ie`(4S) — updated for this release
`ipi`(4S) — new for this release
`is`(4S) — new for this release
`mcp`(4S) — updated for this release
`mtio`(4) — updated for this release
`sd`(4S) — updated for this release
`st`(4S) — updated for this release

Section 8

`config`(8) — updated for this release
`eeprom`(8) — updated for this release
`iostat`(8) — updated for this release
`vmstat`(8) — updated for this release

Instructions

Unchanged manual pages are fully reprinted here when any part appears on the same sheet of paper as new or updated manual pages. This allows you to exchange pages from the *SunOS Reference Manual* without removing any information that appears on the same sheet as an updated manual page. The page numbers on these insertion pages are consistent with those of the *SunOS Reference Manual.*

To insert these pages:

- Look at the insertion pages, find the first block of consecutively numbered pages.
- Find the same block of pages in the *SunOS Reference Manual.*

- Replace the block of pages in *SunOS Reference Manual* with the block of insertion pages.
- Repeat this procedure for all remaining insertion pages.

For example, the second block of insertion pages is numbered 1223-1232d, which corresponds to 1223-1232 in the *SunOS Reference Manual.* Notice you are inserting more pages than you removed. The new pages mentioned above, `id`(4S), `ipi`(4S), and `is`(4S) are inserted here.

NAME

intro – introduction to device drivers, protocols, and network interfaces

DESCRIPTION

This section describes device drivers, high-speed network interfaces, and protocols available under SunOS. The system provides drivers for a variety of hardware devices, such as disks, magnetic tapes, serial communication lines, mice and frame buffers, as well as virtual devices such as pseudo-terminals and windows. SunOS provides hardware support and a network interface for the 10-Megabit Ethernet, along with interfaces for the IP protocol family and a STREAMS-based Network Interface Tap (NIT) facility.

In addition to describing device drivers that are supported by the 4.3BSD operating system, this section contains subsections that describe:

- SunOS-specific device drivers, under '4S'.
- Protocol families, under '4F'.
- Protocols and raw interfaces, under '4P'.
- STREAMS modules, under '4M'.
- Network interfaces, under '4N'.

Configuration

The SunOS kernel can be configured to include or omit many of the device drivers described in this section. The CONFIG section of the manual page gives the line(s) to include in the kernel configuration file for each machine architecture on which a device is supported. If no specific architectures are indicated, the configuration syntax applies to all Sun systems.

The GENERIC kernel is the default configuration for SunOS. It contains all of the optional drivers for a given machine architecture. See **config**(8), for details on configuring a new SunOS kernel.

The manual page for a device driver may also include a DIAGNOSTICS section, listing error messages that the driver might produce. Normally, these messages are logged to the appropriate system log using the kernel's standard message-buffering mechanism (see **syslogd**(8)); they may also appear on the system console.

Ioctls

Various special functions, such as querying or altering the operating characteristics of a device, are performed by supplying appropriate parameters to the **ioctl**(2) system call. These parameters are often referred to as "ioctls." Ioctls for a specific device are presented in the manual page for that device. Ioctls that pertain to a class of devices are listed in a manual page with a name that suggests the class of device, and ending in 'io', such as **mtio**(4) for magnetic tape devices, or **dkio**(4S) for disk controllers. In addition, some ioctls operate directly on higher-level objects such as files, terminals, sockets, and streams:

- Ioctls that operate directly on files, file descriptors, and sockets are described in **filio**(4). Note: the **fcntl**(2) system call is the primary method for operating on file descriptors as such, rather than on the underlying files. Also note that the **setsockopt** system call (see **getsockopt**(2)) is the primary method for operating on sockets as such, rather than on the underlying protocol or network interface. Ioctls for a specific network interface are documented in the manual page for that interface.
- Ioctls for terminals, including pseudo-terminals, are described in **termio**(4). This manual page includes information about both the BSD **termios** structure, as well as the System V **termio** structure.
- Ioctls for STREAMS are described in **streamio**(4).

Devices Always Present

Device drivers present in every kernel include:

- The paging device; see **drum**(4).
- Drivers for accessing physical, virtual, and I/O space in memory; see **mem**(4S).
- The data sink; see **null**(4).

Terminals and Serial Communications Devices

Serial communication lines are normally supported by the terminal driver; see **tty**(4). This driver manages serial lines provided by communications drivers, such as those described in **mti**(4S) and **zs**(4S). The terminal driver also handles serial lines provided by virtual terminals, such as the Sun console monitor described in **console**(4S), and true pseudo-terminals, described in **pty**(4).

Disk Devices

Drivers for the following disk controllers provide standard block and raw interfaces under SunOS;

- SCSI controllers, in **sd**(4S),
- Xylogics 450 and 451 SMD controllers, in **xy**(4S),
- Xylogics 7053 SMD controllers, in **xd**(4S).

Ioctls to query or set a disk's geometry and partitioning are described in **dkio**(4S).

Magnetic Tape Devices

Magnetic tape devices supported by SunOS include those described in **ar**(4S), **tm**(4S), **st**(4S), and **xt**(4S). Ioctls for all tape-device drivers are described in **mtio**(4S).

Frame Buffers

Frame buffer devices include color frame buffers described in the **cg***(4S) manual pages, monochrome frame buffers described in the **bw***(4S) manual pages, graphics processor interfaces described in the **gp***(4S) manual pages, and an indirect device for the console frame buffer described in **fb**(4S). Ioctls for all frame-buffer devices are described in **fbio**(4S).

Miscellaneous Devices

Miscellaneous devices include the console keyboard described in **kbd**(4S), the console mouse described in **mouse**(4S), window devices described in **win**(4S), and the DES encryption-chip interface described in **des**(4S).

Network-Interface Devices

SunOS supports the 10-Megabit Ethernet as its primary network interface; see **ec**(4S), **ie**(4S), and **le**(4S) for details. However, a software loopback interface, **lo**(4) is also supported. General properties of these network interfaces are described in **if**(4N), along with the ioctls that operate on them.

Support for network routing is described in **routing**(4N).

Protocols and Protocol Families

SunOS supports both socket-based and STREAMS-based network communications. The Internet protocol family, described in **inet**(4F), is the primary protocol family primary supported by SunOS, although the system can support a number of others. The raw interface provides low-level services, such as packet fragmentation and reassembly, routing, addressing, and basic transport for socket-based implementations. Facilities for communicating using an Internet-family protocol are generally accessed by specifying the AF_INET address family when binding a socket; see **socket**(2) for details.

Major protocols in the Internet family include:

- The Internet Protocol (IP) itself, which supports the universal datagram format, as described in **ip**(4P). This is the default protocol for SOCK_RAW type sockets within the AF_INET domain.
- The Transmission Control Protocol (TCP); see **tcp**(4P). This is the default protocol for SOCK_STREAM type sockets.
- The User Datagram Protocol (UDP); see **udp**(4P). This is the default protocol for SOCK_DGRAM type sockets.
- The Address Resolution Protocol (ARP); see **arp**(4P).
- The Internet Control Message Protocol (ICMP); see **icmp**(4P).

The Network Interface Tap (NIT) protocol, described in **nit**(4P), is a STREAMS-based facility for accessing the network at the link level.

SEE ALSO

fcntl(2), **getsockopt**(2), **ioctl**(2), **socket**(2), **ar**(4S), **arp**(4P), **dkio**(4S), **drum**(4), **ec**(4S), **fb**(4S), **fbio**(4S), **filio**(4), **icmp**(4P), **if**(4N), **inet**(4F), **ip**(4P), **kbd**(4S), **le**(4), **lo**(4), **mbio**(4S), **mem**(4S), **mti**(4), **mtio**(4), **nit**(4P), **null**(4), **pty**(4), **routing**(4N), **sd**(4S), **st**(4S) **streamio**(4), **tcp**(4P), **termio**(4), **tm**(4S), **tty**(4), **udp**(4P), **win**(4S), **xd**(4S), **xy**(4S), **zs**(4S)

LIST OF DEVICES, INTERFACES AND PROTOCOLS

Name	Appears on Page	Description
alm	**mcp**(4S)	Asynchronous Line Multiplexer
ar	**ar**(4S)	Archive 1/4 inch Streaming Tape Drive
arp	**arp**(4P)	Address Resolution Protocol
bk	**bk**(4)	line discipline for machine-machine communication
bwone	**bwone**(4S)	Sun-1 black and white frame buffer
bwtwo	**bwtwo**(4S)	Sun-3/Sun-2 black and white frame buffer
cgeight	**cgeight**(4S)	24-bitcolor memory frame buffer
cgfour	**cgfour**(4S)	Sun-3 color memory frame buffer
cgone	**cgone**(4S)	Sun-1 color graphics interface
cgsix	**cgsix**(4S)	Sun-3, Sun-4, and Sun-3x low-end graphics accelerator
cgthree	**cgthree**(4S)	Sun386i color memory frame buffer
cgtwo	**cgtwo**(4S)	Sun-3/Sun-2 color graphics interface
clone	**clone**(4)	open any minor device on a STREAMS driver
console	**console**(4S)	console driver and terminal emulator for the Sun workstation
db	**db**(4M)	SunDials STREAMS module
des	**des**(4S)	DES encryption chip interface
dkio	**dkio**(4S)	generic disk control operations
drum	**drum**(4)	paging device
ec	**ec**(4S)	3Com 10 Mb/s Ethernet interface
fb	**fb**(4S)	driver for Sun console frame buffer
fbio	**fbio**(4S)	general properties of frame buffers
fd	**fd**(4S)	Disk driver for Floppy Disk Controllers
filio	**filio**(4)	ioctls that operate directly on files, file descriptors, and sockets
fpa	**fpa**(4S)	Sun-3 floating point accelerator
gpone	**gpone**(4S)	Sun-3/Sun-2 graphics processor
icmp	**icmp**(4P)	Internet Control Message Protocol
id	**id**(4S)	disk driver for IPI disk controllers
ie	**ie**(4S)	Intel 10 Mb/s Ethernet interface
if	**if**(4N)	general properties of network interfaces
inet	**inet**(4F)	Internet protocol family
ip	**ip**(4P)	Internet Protocol
ipi	**ipi**(4S)	IPI driver
is	**is**(4S)	IPI channel driver for Sun IPI string controllers
kb	**kb**(4M)	Sun keyboard STREAMS module
kbd	**kbd**(4S)	Sun keyboard
kmem	**mem**(4S)	main memory and bus I/O space
ldterm	**ldterm**(4M)	standard terminal STREAMS module
le	**le**(4S)	Sun-3/50, Sun-3/60 10MB Ethernet interface
lo	**lo**(4)	software loopback network interface
lofs	**lofs**(4S)	loopback virtual file system
mbio	**mem**(4S)	main memory and bus I/O space
mbmem	**mem**(4S)	main memory and bus I/O space
mcp	**mcp**(4S)	MCP Multiprotocol Communications Processor

mem	**mem**(4S)	main memory and bus I/O space
mouse	**mouse**(4S)	Sun mouse
ms3	**mouse**(4S)	Sun mouse
ms	**ms**(4M)	Sun mouse STREAMS module
mti	**mti**(4S)	Systech MTI-800/1600 multi-terminal interface
mtio	**mtio**(4)	UNIX system magnetic tape interface
NFS	**nfs**(4P)	network file system
nif_pf	**nit_pf**(4M)	streams NIT packet filtering module
nit	**nit**(4P)	Network Interface Tap facility
nit_buf	**nit_buf**(4M)	streams NIT buffering module
nit_if	**nit_if**(4M)	streams NIT device interface module
null	**null**(4)	data sink
pp	**pp**(4)	Centronics-compatible parallel printer port
pty	**pty**(4)	pseudo terminal driver
root	**root**(4S)	pseudo-driver for Sun root disk
routing	**routing**(4N)	system supporting for local network packet routing
sd	**sd**(4S)	Disk driver for SCSI Disk Controllers
sockio	**sockio**(4)	ioctls that operate directly on sockets
st	**st**(4S)	Sysgen SC 4000 and Emulex MT-02 Tape Controller
streamio	**streamio**(4)	STREAMS ioctl commands
tcp	**tcp**(4P)	Transmission Control Protocol
termio	**termio**(4)	general terminal interface
tm	**tm**(4S)	tapemaster 1/2 inch tape drive
ttcompat	**ttcompat**(4M)	V7/4BSD compatibility STREAMS module
tty	**tty**(4)	controlling terminal interface
udp	**udp**(4P)	User Datagram Protocol
vme16d16	**mem**(4S)	main memory and bus I/O space
vme16d32	**mem**(4S)	main memory and bus I/O space
vme24d16	**mem**(4S)	main memory and bus I/O space
vme24d32	**mem**(4S)	main memory and bus I/O space
vme32d16	**mem**(4S)	main memory and bus I/O space
vme32d32	**mem**(4S)	main memory and bus I/O space
vp	**vp**(4S)	Ikon 10071-5 Versatec parallel printer interface
vpc	**vpc**(4S)	Systech VPC-2200 Versatec plotter and Centronics printer
win	**win**(4S)	Sun window system
xd	**xd**(4S)	Disk driver for Xylogics 7053 SMD Disk Controller
xt	**xt**(4S)	Xylogics 472 1/2 inch tape controller
xy	**xy**(4S)	Disk driver for Xylogics SMD Disk Controllers
zero	**zero**(4S)	source of zeroes
zs	**zs**(4S)	Zilog 8530 SCC serial comunications driver

NAME

dkio – generic disk control operations

DESCRIPTION

All Sun disk drivers support a set of ioctl's for disk formatting and labeling operations. Basic to these ioctl's are the definitions in **<sun/dkio.h>**:

```
/*
* Structures and definitions for disk io control commands
*/
/* Disk identification */
struct dk_info {
        int     dki_ctlr;               /* controller address */
        short   dki_unit;               /* unit (slave) address */
        short   dki_ctype;              /* controller type */
        short   dki_flags;              /* flags */
};
/* controller types */
#define DKC_UNKNOWN             0
#define DKC_SMD2180             1
#define DKC_DSD5215             5
#define DKC_XY450               6
#define DKC_ACB4000             7
#define DKC_MD21                8
#define DKC_CSS         12
#define DKC_NEC765      13      /* floppy on Sun386i */
/* flags */
#define DKI_BAD144      0x01    /* use DEC std 144 bad sector fwding */
#define DKI_MAPTRK      0x02    /* controller does track mapping */
#define DKI_FMTTRK      0x04    /* formats only full track at a time */
#define DKI_FMTVOL      0x08    /* formats only full volume at a time */
/* Definition of a disk's geometry */
struct dk_geom {
        unsigned short  dkg_ncyl;       /* # of data cylinders */
        unsigned short  dkg_acyl;       /* # of alternate cylinders */
        unsigned short  dkg_bcyl;       /* cyl offset (for fixed head area) */
        unsigned short  dkg_nhead;      /* # of heads */
        unsigned short  dkg_bhead;      /* head offset (for Larks, etc.) */
        unsigned short  dkg_nsect;      /* # of sectors per track */
        unsigned short  dkg_intrlv;     /* interleave factor */
        unsigned short  dkg_gap1;       /* gap 1 size */
        unsigned short  dkg_gap2;       /* gap 2 size */
        unsigned short  dkg_apc;        /* alternates per cyl (SCSI only) */
        unsigned short  dkg_extra[9];   /* for compatible expansion */
};
/* disk io control commands */
#define DKIOCGGEOM  _IOR(d, 2, struct dk_geom)      /* Get geometry */
#define DKIOCSGEOM          _IOW(d, 3, struct dk_geom)      /* Set geometry */
#define DKIOCGPART          _IOR(d, 4, struct dk_map)       /* Get partition info */
#define DKIOCSPART          _IOW(d, 5, struct dk_map)       /* Set partition info */
#define DKIOCINFO           _IOR(d, 8, struct dk_info)      /* Get info */
#define DKIOCWCHK           _IOWR(d, 115, int)      /* Toggle write check */
```

The **DKIOCINFO** ioctl returns a **dk_info** structure which tells the type of the controller and attributes about how bad-block processing is done on the controller. The **DKIOCGPART** and **DKIOCSPART** get and set the controller's current notion of the partition table for the disk (without changing the partition table on the disk itself), while the **DKIOCGGEOM** and **DKIOCSGEOM** ioctl's do similar things for the per-drive geometry information. The **DKIOCWCHK** enables or disables a disk's write check capabilities.

SEE ALSO

ip(4P), **sd**(4S), **xy**(4S), **dkctl**(8)

NAME

icmp – Internet Control Message Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip_icmp.h>

s = socket(AF_INET, SOCK_RAW, proto);
```

DESCRIPTION

ICMP is the error and control message protocol used by the Internet protocol family. It is used by the kernel to handle and report errors in protocol processing. It may also be accessed through a "raw socket" for network monitoring and diagnostic functions. The protocol number for ICMP, used in the *proto* parameter to the socket call, can be obtained from **getprotobyname** (see **getprotoent**(3N)). ICMP sockets are connectionless, and are normally used with the *sendto* and *recvfrom* calls, though the **connect**(2) call may also be used to fix the destination for future packets (in which case the **read**(2V) or **recv**(2) and **write**(2V) or **send**(2) system calls may be used).

Outgoing packets automatically have an Internet Protocol (IP) header prepended to them. Incoming packets are provided to the holder of a raw socket with the IP header and options intact.

ICMP is an unreliable datagram protocol layered above IP. It is used internally by the protcol code for various purposes including routing, fault isolation, and congestion control. Receipt of an ICMP "redirect" message will add a new entry in the routing table, or modify an existing one. ICMP messages are routinely sent by the protocol code. Received ICMP messages may be reflected back to users of higher-level protocols such as TCP or UDP as error returns from system calls. A copy of all ICMP message received by the system is provided using the ICMP raw socket.

ERRORS

A socket operation may fail with one of the following errors returned:

EISCONN	when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected;
ENOTCONN	when trying to send a datagram, but no destination address is specified, and the socket hasn't been connected;
ENOBUFS	when the system runs out of memory for an internal data structure;
EADDRNOTAVAIL	when an attempt is made to create a socket with a network address for which no network interface exists.

SEE ALSO

connect(2), **read**(2V), **recv**(2), **send**(2), **write**(2V), **getprotoent**(3N), **inet**(4F), **ip**(4P), **routing**(4N)

Postel, Jon, *Internet Control Message Protocol — DARPA Internet Program Protocol Specification*, RFC 792, Network Information Center, SRI International, Menlo Park, Calif., September 1981. (Sun 800-1064-01)

BUGS

Replies to ICMP "echo" messages which are source routed are not sent back using inverted source routes, but rather go back through the normal routing mechanisms.

NAME

id – disk driver for IPI disk controllers

CONFIG — SUN-3 and SUN-4 SYSTEMS

```
controller      idc0     at ipi ? csr 0x0000ff priority 2  # channel 0 slave 0
disk            id 0x000 at idc0 drive 0   # facility 0
disk            id 0x001 at idc0 drive 1
disk            id 0x002 at idc0 drive 2
disk            id 0x003 at idc0 drive 3
disk            id 0x004 at idc0 drive 4
disk            id 0x005 at idc0 drive 5
disk            id 0x006 at idc0 drive 6
disk            id 0x007 at idc0 drive 7

controller      idc1     at ipi ? csr 0x0001ff priority 2 # channel 0 slave 1
disk            id 0x010 at idc1 drive 0   # facility 0
disk            id 0x011 at idc1 drive 1
disk            id 0x012 at idc1 drive 2
disk            id 0x013 at idc1 drive 3
disk            id 0x014 at idc1 drive 4
disk            id 0x015 at idc1 drive 5
disk            id 0x016 at idc1 drive 6
disk            id 0x017 at idc1 drive 7

controller      idc2     at ipi ? csr 0x0002ff priority 2 # channel 0 slave 2
disk            id 0x020 at idc2 drive 0   # facility 0
disk            id 0x021 at idc2 drive 1
disk            id 0x022 at idc2 drive 2
disk            id 0x023 at idc2 drive 3
disk            id 0x024 at idc2 drive 4
disk            id 0x025 at idc2 drive 5
disk            id 0x026 at idc2 drive 6
disk            id 0x027 at idc2 drive 7

controller      idc3     at ipi ? csr 0x0003ff priority 2 # channel 0 slave 3
disk            id 0x030 at idc3 drive 0   # facility 0
disk            id 0x031 at idc3 drive 1
disk            id 0x032 at idc3 drive 2
disk            id 0x033 at idc3 drive 3
disk            id 0x034 at idc3 drive 4
disk            id 0x035 at idc3 drive 5
disk            id 0x036 at idc3 drive 6
disk            id 0x037 at idc3 drive 7
```

The four **controller** lines given in the synopsis section above correspond to the first, second, third, and fourth IPI disk controller in a Sun system. These controllers are on IPI channels or combined on integrated channel/disk-controller cards such as the ISP-80 IPI disk controller. The **csr** value gives the channel, slave, and facility address of the controller. The facility address should always be **0xff**. See is(4S).

DESCRIPTION

In order to accommodate a large number of disk drives, a multiple major number device addressing scheme is used. The minor number is formed using the low order 5 bits of the unit number and the 3-bit partition number. The low-order 3 bits of the minor number gives the partition number on the drive. The high order bits of the unit number are added to the first major number for **id** devices to give the major number of the particular device. The unit number itself is formed as follows: the 4-bit facility number form the low order 4-bits, next the 3-bit slave number, and then the channel number.

The standard device names begin with **id** followed by three hex digits giving the channel number, slave number, and facility number, and then a letter **a-h**, for partitions 0-7 respectively.

The block files access the disk using the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a *raw* interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call usually results in only one I/O operation; therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files conventionally begin with an extra '**r**'.

In raw I/O, counts should be a multiple of 512 bytes (a disk sector). Likewise **directory**(3) calls should specify a multiple of 512 bytes. Depending on the channel adaptor, the buffer for raw reads or writes may be required to be on a 2-byte or 4-byte boundary.

The **ioctl**( ) interface described in **dkio**(4S) is supported by this driver. The DKIOCSCMD ioctl can be used to issue certain IPI commands to the drive. The argument structure is:

```
struct dk_cmd {
        u_short dkc_cmd;        /* command to be executed */
        int     dkc_flags;      /* execution flags */
        daddr_t dkc_blkno;      /* disk address for command */
        int     dkc_secnt;      /* sector count for command */
        caddr_t dkc_bufaddr;    /* user's buffer address */
        u_int   dkc_buflen;     /* size of user's buffer */
};
```

The lower 8-bits of the **dkc_cmd** field indicate one of the supported commands listed below. The upper 8-bits indicate the IPI Opcode modifier. These commands are defined in **/usr/include/sundev/ipi3.h**. Block numbers are not remapped by the partition map when these commands are used. The supported commands are:

IP_READ
IP_WRITE

Read or write data. The addressing is always by logical block (ignoring [**a-h**] logical partition information), the Opcode modifier is ignored.

IP_READ_DEFLIST
IP_WRITE_DEFLIST

Read or write one of the defect lists. The defect list is selected by the Opcode modifier in bits <15:8> of the **dkc_cmd**.

IP_FORMAT

Format a range of cylinders. For this command, the block number and sector count fields must both be a multiple of the number of blocks per cylinder. The **dk_buflen** field must be zero for this command.

IP_REALLOC

Reallocate a block. The controller attempts to recover the data from the old block being reallocated. If the old data cannot be recovered, a conditional success status is presented and a message may be printed. The **dk_buflen** field must be zero for this command.

DISK SUPPORT

This driver handles all supported IPI drives by reading controller attributes and a label from sector 0 of the drive which describes the disk geometry and partitioning.

The **id000a** partition is normally used for the root file system on a disk, the **id0??b** partition as a paging area, and the **id???c** partition for pack-pack copying (it normally maps the entire disk). The **id???h** partition is generally a partition suitable for holding the **/usr** file system. The rest of the disk is normally the **id???h** partition.

FILES

/dev/id[0-7][0-7][0-7][a-h]	block files
/dev/rid[0-7][0-7][0-7][a-h]	raw files

SEE ALSO

lseek(2), **read**(2V), **write**(2V), **directory**(3), **dkio**(4S) **is**(4S)

DIAGNOSTICS

The following messages are usually preceded by either **idc***n*, indicating a message associated with controller *n*, or **id***csf*, indicating a message associated with the drive *csf*, where *csf* is the drive's channel, slave, and facility address.

idc*n*: **configured with bad IPI address** *addr*
The IPI address of the controller should end with '**ff**'.

idc*n*: **unknown ctlr vendor id: manuf** '*name*' **model** '*model*'
The controller type is unsupported. The driver has not been tested with the controller being used.

idc*n*: **unknown string ctlr type**
The controller type is unsupported. The driver has not been tested with the controller being used.

id*csf*: **out of memory for label info**
The disk could not be initialized because the driver could not allocate kernel memory for the label.

id*csf*: **out of memory for label info**
The disk could not be initialized because the driver could not allocate kernel memory for the label.

id*csf*: **warning: starting block address nonzero:** *addr*
The disk attributes indicate a non-zero first block number. None of the supported controllers do this.

id*csf*: **zero blocks per track**
The disk attributes indicate zero blocks per track. None of the supported controllers do this.

id*csf*: **inconsistent geometry: blk/cyl** *num* **blk/trk** *num* **head** *num*
The disk attributes indicate different number of blocks per cylinder than would be expected from the blocks per track and the number of tracks (heads).

id*csf*: **inconsistent geometry: tot blks** *num* **blks/cyl** *num* **pcyl** *num*
The disk attributes indicate different number of total blocks than would be expected from the blocks per cylinder and the number of cylinders.

id*csf*: **invalid logical block size** *num*
The disk logical block size, given by the logical block size attribute parameter, is not a power of two less than 2**31.

idintr: bad rupt NULL q_dev
The interrupt routine was called with a request that cannot be identified.

idc*n*: **ctlr message:** '*message-text*'
The firmware in the disk controller issued the message *message-text*. The controller manual should describe the message.

idc*n*: **ctlr failure fru** *num* **message:** '*message-text*'
The firmware in the disk controller issued the failure message *message-text*, and indicates a likely field replaceable unit number (printed in hex). The controller manual should describe the message and FRU number.

id*csf*: **corrupt label**
The label checksum mismatched.

id*csf*: <*ASCII-label*>
The label contains an ASCII string which is printed after the label is read. The ASCII string should give the disk type and number of cylinders, heads, and sectors.

id*csf*: warning: label doesn't match IPI attribute info
id*csf*: label ncyl *num* pcyl *num* acyl *num* head *num* sect *num* rpm *num*
id*csf*: geom ncyl *num* pcyl *num* acyl *num* head *num* sect *num* rpm *num*
Some of the geometry information from the disk label does not match the geometry information from the controller attributes. This may or may not be a problem, but indicates a non-standard label.

id*csf*: id_rdwr: block *num* past end of disk
An **ioctl()** read or write was issued to a block in a cylinder with a number greater than or equal to the number of physical cylinders indicated by the disk geometry from the label or attributes.

id*csf*: id_format failed. result *num*
A format operation failed. The code number printed, *num*, is defined in **/usr/include/sundev/ipi_driver.h.**

id*csf*: id_rdwr_deflist failed. result *num*
A read or write defect list operation failed. The code number printed, *num*, is defined in **/usr/include/sundev/ipi_driver.h.**

id*csf*: id_realloc failed. result *num*
A reallocation operation failed. The code number printed, *num*, is defined in **/usr/include/sundev/ipi_driver.h.**

id*csf*[a-h]: *opcode* block *rel-block* (*block* abs) idintr: bad rupt NULL qdev
An interrupt occurred for which the associated disk could not be located.

id*csf*[a-h]: *opcode* block *rel-block* (*block* abs) idintr: neg cmd_q_len
The driver's count of commands outstanding to the drive became less than zero. This should be recoverable, but indicates a minor driver problem.

id*csf*[a-h]: *opcode* block *rel-block* (*block* abs) poll timeout
A command hung or took longer than the driver expected.

id*csf*[a-h]: *opcode* block *rel-block* (*block* abs) id_cmd_intr: ctlr cmd_q_len negative
The driver's count of commands outstanding to the controller became less than zero. This should be recoverable, but indicates a minor driver problem.

id*csf*[a-h]: *opcode* block *rel-block* (*block* abs) id_error: slave IML not supported
An error response from the controller indicates that it requires an initial microcode load. If this message occurs, controllers which issue this error are not supported.

id*csf*[a-h]: *opcode* block *rel-block* (*block* abs) id_error: slave reset not supported
An error response from the controller indicates that it requires a reset. If this message occurs, controllers which issue this error are not supported.

id*csf*[a-h]: *opcode* block *rel-block* (*block* abs) id_error: full queue: not handled
An error response from the controller indicates that its internal command queue is full, and the command cannot be accepted. This message should not occur because the driver respects the queue limits determined from the controller attributes.

id*csf*[a-h]: *opcode* block *rel-block* (*block* abs) id_error: log read not supported
The controller presented status indicating that it's internal log has overflowed and should be read by the host. If this message occurs, controllers which issue this status are not supported.

id*csf*[a-h]: *opcode* block *rel-block* (*block* abs) id_error: aborted command not supported
An interrupt indicated that the associated command had been terminated by abort. If this message occurs, the driver does not issue aborts or expect this status.

id*csf*[a-h]: *opcode* block *rel-block* (*block* abs) id_error_parse: no response where expected
An interrupt occured which should have been accompanied by a response packet, but none was found.

id*csf*[a-h]: *opcode* block *rel-block* (*block* abs) missing interrupt – attempting recovery

The command took much longer than expected, indicating a possible problem with the controller or drive. Recovery of the controller will be started, and the command will be retried if possible.

id*csf*[a-h]: *opcode* block *rel-block* (*block* abs) missing interrupt – recovery in progress
The command took much longer than expected, indicating a possible problem with the controller or drive. Since recovery was already in progress, the command was added to the list or commands to be retried after recovery.

id*csf*[a-h]: *opcode* block *rel-block* (*block* abs) no memory for label
The driver could not obtain a temporary memory area for the disk label.

id*csf*[a-h]: *opcode* block *rel-block* (*block* abs) cannot abort yet
The recovery code cannot abort specific commands. It will assume the command was cleared by a reset and simply reissue the command.

id*csf*: id_recover *state-number state-name*
id*cnum*: id_recover *state-number state-name*
id*csf*: id_recover_intr *state-number state-name* **result** *result*
id*cnum*: id_recover_intr *state-number state-name* **result** *result*
These messages trace the recovery action during missing interrupt handling and controller recovery. These are for diagnosing problems in the event that recovery is unsuccessful.

id*cnum*: synchronous command setup failed
During recovery or initialization, the setup for a command failed, probably due to an internal error or lack of free memory.

IPI DIAGNOSTICS

The following diagnostics come directly from controller status. The messages may occur in combinations determined by the controller. In most cases, more information about the conditions under which the status is issued will be found in the controller manual.

In the examples shown, the message for the response parameter is shown first, then the message for a single bit in the response parameter. In practice, there will be a response parameter message followed by one or more messages for the individual bits, all on the same line.

The messages will be preceded by the disk or controller number. The message descriptions refer to the appropriate disk or controller as the addressee, since it was addressed in the command or response.

Normally, these messages are not printed when the condition described arises. They will only be printed if the condition is "interesting." Certain conditions print all of the messages associated with a response, even though the individual messages might not be especially interesting.

The description of the message meaning is excerpted from the IPI-3 standards document. Refer to it for a more complete explanation. [XXX need full reference XXX]

Message/Microcode Exception. Message.
The slave has included a message within Extended Substatus for the master.

Message/Microcode Exception. Failure Message.
The slave encountered a failure condition which resulted in the identification of a FRU (Field Replaceable Unit).

Message/Microcode Exception. Port Disable Pending.
The addressee has received a manual or programmed Port Disable command that will take effect when the Disable conditions are met.

Message/Microcode Exception. Port Response.
A port has executed a Port Response command.

Intervention Required. Not P-Available.
The selected addressee is not powered on or is not installed.

Intervention Required. Not Ready.

The selected addressee cannot execute its intended functions. The addressees' Not Ready condition may be cleared by operator intervention.

Intervention Required. Not P-Available Transition.
This is presented by the controller to advise the driver that a facility has become Not P-Available since the time that a command addressed to it was accepted. Note: if the transition had occurred before the command packet was accepted, the status would have been Not P-Available.

Intervention Required. Not Ready Transition.
This is presented by the controller to advise the driver that a facility has dropped ready since the time that a command addressed to it was accepted. Note: if the transition had occurred before the command packet was accepted, the status would have been Not Ready.

Intervention Required. Attribute Table may be corrupted.
The controller has encountered a condition under which it is possible that the Attributes table has been corrupted, and it is not prepared to continue operation without master intervention.

Intervention Required. Busy.
The command cannot be executed because the addressee has been Busy for a time determined by the Facility Timeout Value specified in Attributes.

Alternate Port Exception. Priority Reserve Issued.
The addressee has been instructed to release allegiance to this port because of a Priority Reserve from an alternate port.

Alternate Port Exception. Attributes Updated.
An Attributes command has been issued from an alternate port which has changed the addressee's attributes.

Alternate Port Exception. Initialization COmpleted.
The addressee has completed an initialization procedure which may have affected this port, and was originated by a Reset from an alternate port.

Alternate Port Exception. Format Completed.
The addressee has completed a **FORMAT** command from an alternate port.

Alternate Port Exception. Facility switched to other port.
The slave has determined that the facility is switched to another port.

Machine Exception. No longer busy.
The addressee is notifing the master that it is no longer busy.

Machine Exception. P-Available.
This is presented asynchronously by the slave to advise the master that a facility which was previously Not P-Available has become P-Available.

Machine Exception. Ready.
This is presented asynchronously by the slave to advise the master that a facility which was not previously ready has become Ready.

Machine Exception. Operation Timeout.
There has been a failure condition in the addressee which has been detected by an internal timeout mechanism.

Machine Exception. Physical Interface Check.
The slave detected a check condition on the Physical Interface, for example, an invalid sequence generated by the "state machine" or parity error on the bus(es).

Machine Exception. Slave Initiated Reset.
An internal condition caused the slave to initiate a reset; the master shall assume all outstanding commands and buffer contents are either lost or suspect.

Machine Exception. Environmental Error.

Some condition internal or external to the addressee has been detected which may cause a failure condition(s), for example, a temperature sensor alert.

Machine Exception. Power Fail Alert.
The addressee has detected an impending power failure condition.

Machine Exception. Data Check (on Raw Data).
The master has requested raw data and the addresseee has detected a data error.

Machine Exception. Uncorrectable Data Check.
The slave detected a data error which has persisted after the slave has exhausted any possible recovery actions. On write operations, the malfunction may have caused invalid data to be recorded.

Machine Exception. Fatal Error.
The addressee detected an internal machine error that precludes execution or continuation of the current command.

Machine Exception. Hardware Write Protected.
An attempt was made to write on a drive that was protected against writing by something physical, for example, a switch on the drive.

Machine Exception. Queue Full.
The command queue for the addressee is full.

Machine Exception. Command Failure.
The command in execution encountered a condition which caused it to complete correctly but unsuccessfully, for example, a COMPARE of two files detected a discrepancy.

Machine Exception. Read Access Violation.
An attempt was made to read on an addressee which had been protected using Access Permits.

Machine Exception. Write Access Violation.
An attempt was made to write on an addressee which had been protected using Access Permits.

Machine Exception. Data Overrun.
This can occur during direct data transfer, or if the slave has a buffer which was not adequate, and the buffer overran during a read or a write operation.

Machine Exception. Reallocation space exhausted.
Space required for reallocation of data due to media defects is not available, that is, all space assigned for that purpose has been exhausted.

Machine Exception. Unexpected Master Action.
The slave has encountered an unexpected action by the master, for example, the Master Status at the Physical Interface does not correlate to the anticipated status, no status was expected and some was presented by the master, the master did not respond with Data I/O or Control of Bus following a Transfer Notification packet.

Machine Exception. Error Log Full.
The Error Log capacity has been exceeded.

Machine Exception. Defect Directory Full.
The Defect Directory capacity has been exceeded and no more blocks can be re-allocated.

Command Exception. Invalid Packet Length.
The packet length is invalid, for example, the length of the parameter list plus the basic packet does not equal the packet length.

Command Exception. Invalid Command Reference Number.
The Command Reference Number is invalid, for example, it duplicates one in a command that is currently active.

Command Exception. Invalid Slave Address.

The Slave Address in the command packet is invalid, for example, it does not match the selected slave's address.

Command Exception. Invalid Facility Address.
The Facility Address in the command packet is invalid.

Command Exception. Invalid Selection Address.
The facility selected at the Physical Interface does not match the facility address supplied in the command packet.

Command Exception. Invalid Opcode.
The command packet contained an invalid or unsupported Opcode.

Command Exception. Invalid Modifier.
The Modifier was invalid or is not supported for the Opcode specified.

Command Exception. Invalid Extent.
The Data Address (for example, the block number) plus the Count specified in an Extent parameter is not valid for the addressee.

Command Exception. Out of Context.
The slave has encountered a situation in which it cannot process the command because it considers it out of context for example, a **RESUME** command without a previous **COPY.**

Command Exception. Invalid Parameter.
One or more of the parameters in the command packet was invalid.

Command Exception. Missing Parameter.
One or more of the parameters required in the command is not present.

Command Exception. Reserved Value nonzero.
A reserved value defined by the standard does not contain zero.

Command Exception. Invalid Combination.
The addressee has detected that two valid but mutually exclusive options have been selected by the master.

Command Aborted. Command Aborted.
The command this response packet is related to was **ABORT**ed by the master.

Command Aborted. Command Sequence Terminated.
Command Sequencing was terminated because this command failed to complete successfully.

Command Aborted. Unexecuted Command from Terminated Sequence.
The command related to this response packet was not executed but was terminated because a prior command which was sequenced to it failed to complete successfully.

Command Aborted. Command Chain Terminated.
Command Chaining was terminated because this command failed to complete successfully.

Command Aborted. Unexecuted Command from Terminated Chain.
The command related to this response packet was not executed but was terminated because a prior command which was chained to it failed to complete successfully.

Command Aborted. Command Order Terminated.
Command Ordering was terminated because this command failed to complete successfully.

Command Aborted. Unexecuted Command from Terminated Order.
The command related to this response packet was not executed but was terminated because a prior command which was ordered to it failed to complete successfully.

Conditional Success. Logging Data Appended.
The slave has appended information in this response which the slave is advising the master is relevant to be logged.

Conditional Success. Abort Received – no Command Active.
An ABORT command was issued to an addressee in the L-Available condition but the referenced command could not be found.

Conditional Success. Abort Received – Status Pending.
An ABORT command was issued to an addressee which has the response status for the referenced command pending, for example, the command has been completed.

Conditional Success. Abort Received – Not Operational.
An ABORT command was issued to a facility which is Not Operational.

Conditional Success. Anticipated Error.
The addressee has detected a condition which may result in future error conditions, for example, on disc seek retries were needed.

Conditional Success. Anticipated Data Error.
The addressee has detected a condition which may result in future data errors, for example, successive retries needed for reading disk data.

Conditional Success. Re-allocation Required.
The addressee has detected a data error condition which requires reallocation action, for example, an unrecoverable read error.

Conditional Success. Re-allocation Discontinuity.
The slave has automatically reallocated a block which contained a data error and the reallocated data is now no longer in close proximity to the blocks previously contiguous to it.

Conditional Success. Defect Directory Threshold Exceeded.
The threshold within the addressee's Defect Directory has been exceeded, indicating that there is a limited number of entries remaining for adding more defects.

Conditional Success. Error Retry Performed.
The addressee has completed the command but error retry had to be invoked. Note: Error Retry does not include actions associated with data transfer.

Conditional Success. Data Retry Performed.
The addressee has completed the command but data retry had to be invoked, for example, a physical re-read. Data Retry includes all actions associated with the transfer of data.

Conditional Success. Motion Retry Performed.
The addressee has completed the command but motion retry had to be invoked.

Conditional Success. Data Correction Performed.
The addressee has completed the command but data correction had to be applied.

Conditional Success. Soft Error.
The slave detected an internal machine error that did not preclude execution or continuation of the current command.

Conditional Success. Release of Unreserved Addressee.
The addressee has received a release command for which there is no reservation.

Conditional Success. Request Diagnostic Control Command.
As a result of executing a diagnostic command which provided more information than can be returned by a Response, the addressee is requesting that the master issue a Diagnostic Read Command.

Conditional Success. Error Log Request.
The master is requested to capture the contents of the Error Log (which contains manufacturer dependent information) because the threshold has been exceeded.

Conditional Success. Statistics Update Requested.
There has been a change in meaningful statistics during the execution of this command, and the master is requested to update its Statistics Table (if any).

Conditional Success. Parameter Update Requested.
There has been a change in meaningful device parameters during the execution of this command, and the master is requested to update its Statistics Table (if any).

Conditional Success. Asynchronous Event Occurrence.
An asychronous event has occurred which may be described further in Extended Status.

Conditional Success. Master Terminated Transfer.
The previous Information Transfer which had a Master Termination Parameter, was terminated by the master.

Incomplete. Command May be Resumed.
This status is used to advise the master that an otherwise successful command did not complete. The incomplete command remains on the slave's queue, and its Command Reference Number shall remain valid, until the command is resumed or aborted.

Incomplete. Response Packet Truncated.
The maximum Information Transfer Length specified by the attributes was exceeded by the response packet, which was truncated at that size, and the response is considered complete by the slave.

Incomplete. Data Length Difference.
The addressee has not transferred all the information specified in a transfer command.

NAME

ie – Intel 10 Mb/s Ethernet interface

CONFIG — SUN-4 SYSTEM

device ie0 at obio ? csr 0x6000000 priority 3
device ie1 at vme24d16 ? csr 0xe88000 priority 3 vector ieintr 0x75
device ie2 at vme24d16 ? csr 0x31ff02 priority 3 vector ieintr 0x76
device ie3 at vme24d16 ? csr 0x35ff02 priority 3 vector ieintr 0x77

CONFIG — SUN-3x SYSTEM

device ie0 at obio ? csr 0x65000000 priority 3
device ie1 at vme24d16 ? csr 0xe88000 priority 3 vector ieintr 0x75

CONFIG — SUN-3 SYSTEM

device ie0 at obio ? csr 0xc0000 priority 3
device ie1 at vme24d16 ? csr 0xe88000 priority 3 vector ieintr 0x75
device ie0 at vme24d16 ? csr 0x31ff02 priority 3 vector ieintr 0x74

CONFIG — SUN-2 SYSTEM

device ie0 at obio 2 csr 0x7f0800 priority 3
device ie1 at vme24 ? csr 0xe88000 priority 3 vector ieintr 0x75
device ie0 at mbmem ? csr 0x88000 priority 3
device ie1 at mbmem ? csr 0x8c000 flags 2 priority 3

CONFIG — SUN386I SYSTEM

device ie0 at obmem ? csr 0xD0000000 irq 21 priority 3

DESCRIPTION

The **ie** interface provides access to a 10 Mb/s Ethernet network through the Intel 82586 controller chip. For a general description of network interfaces see **if**(4N).

In the Sun-4 lines above, the first line specifies a CPU-board-resident Intel Ethernet interface. The second line specifies a Multibus Intel Ethernet interface for use with VME adapters. The third and fourth line, ie2 and ie3, specify a VME Ethernet expansion option called the Sun-3/E Ethernet expansion board.

In the Sun-3x lines above, the first line specifies a CPU-board-resident Intel Ethernet interface. The second line specifies a Multibus Intel Ethernet interfaces for use with VME adapters.

In the Sun-3 lines above, the first line specifies the CPU-board-resident Intel Ethernet interface. The second line specifies a Multibus Intel Ethernet interface for use with a VME adapter. The third line specifies the Intel Ethernet interface present on a Sun-3 Eurocard board.

In the Sun-2 lines above, the first line specifies the CPU-board-resident Intel Ethernet interface on a Sun-2/50 or Sun-2/160 system. The second line specifies a Multibus Intel Ethernet controller for use with a VME adapter on these systems. The third line specifies the first Multibus Intel Ethernet controller for a Sun-2/120 or Sun-2/170 system. The fourth line specifies the second such controller for these systems.

The Sun386i line above specifies the CPU-board-resident Intel Ethernet interface.

SEE ALSO

if(4N)

DIAGNOSTICS

There are too many driver messages to list them all individually here. Some of the more common messages and their meanings follow.

ie%d: Ethernet jammed

Network activity has become so intense that sixteen successive transmission attempts failed, and the 82586 gave up on the current packet. Another possible cause of this message is a noise source somewhere in the network, such as a loose transceiver connection.

ie%d: no carrier

The 82586 has lost input to its carrier detect pin while trying to transmit a packet, causing the

packet to be dropped. Possible causes include an open circuit somewhere in the network and noise on the carrier detect line from the transceiver.

ie%d: lost interrupt: resetting
The driver and 82586 chip have lost synchronization with each other. The driver recovers by resetting itself and the chip.

ie%d: iebark reset
The 82586 failed to complete a watchdog timeout command in the allotted time. The driver recovers by resetting itself and the chip.

ie%d: WARNING: requeueing
The driver has run out of resources while getting a packet ready to transmit. The packet is put back on the output queue for retransmission after more resources become available.

ie%d: panic: scb overwritten
The driver has discovered that memory that should remain unchanged after initialization has become corrupted. This error usually is a symptom of a bad 82586 chip.

ie%d: giant packet
Provided that all stations on the Ethernet are operating according to the Ethernet specification, this error "should never happen," since the driver allocates its receive buffers to be large enough to hold packets of the largest permitted size. The most likely cause of this message is that some other station on the net is transmitting packets whose lengths exceed the maximum permitted for Ethernet.

NAME

if – general properties of network interfaces

DESCRIPTION

Each network interface in a system corresponds to a path through which messages may be sent and received. A network interface usually has a hardware device associated with it, though certain interfaces such as the loopback interface, **lo**(4), do not.

At boot time, each interface with underlying hardware support makes itself known to the system during the autoconfiguration process. Once the interface has acquired its address, it is expected to install a routing table entry so that messages can be routed through it. Most interfaces require some part of their address specified with an **SIOCSIFADDR** IOCTL before they will allow traffic to flow through them. On interfaces where the network-link layer address mapping is static, only the network number is taken from the ioctl; the remainder is found in a hardware specific manner. On interfaces which provide dynamic network-link layer address mapping facilities (for example, 10Mb/s Ethernets using **arp**(4P),), the entire address specified in the ioctl is used.

The following **ioctl** calls may be used to manipulate network interfaces. Unless specified otherwise, the request takes an **ifreq** structure as its parameter. This structure has the form

```
struct  ifreq {
        char    ifr_name[16];           /* name of interface (e.g. "ec0") */
        union {
                struct  sockaddr ifru_addr;
                struct  sockaddr ifru_dstaddr;
                short   ifru_flags;
        } ifr_ifru;
#define ifr_addr                ifr_ifru.ifru_addr      /* address */
#define ifr_dstaddr     ifr_ifru.ifru_dstaddr   /* other end of p-to-p link */
#define ifr_flags               ifr_ifru.ifru_flags     /* flags */
};
```

SIOCSIFADDR Set interface address. Following the address assignment, the "initialization" routine for the interface is called.

SIOCGIFADDR Get interface address.

SIOCSIFDSTADDR Set point to point address for interface.

SIOCGIFDSTADDR Get point to point address for interface.

SIOCSIFFLAGS Set interface flags field. If the interface is marked down, any processes currently routing packets through the interface are notified.

SIOCGIFFLAGS Get interface flags.

SIOCGIFCONF Get interface configuration list. This request takes an **ifconf** structure (see below) as a value-result parameter. The **ifc_len** field should be initially set to the size of the buffer pointed to by **ifc_buf**. On return it will contain the length, in bytes, of the configuration list.

```
/*
* Structure used in SIOCGIFCONF request.
* Used to retrieve interface configuration
* for machine (useful for programs which
* must know all networks accessible).
*/
struct  ifconf {
        int     ifc_len;        /* size of associated buffer */
        union {
                caddr_t ifcu_buf;
                struct  ifreq *ifcu_req;
        } ifc_ifcu;
#define ifc_buf  ifc_ifcu.ifcu_buf /* buffer address */
#define ifc_req  ifc_ifcu.ifcu_req /* array of structures returned */
};
```

SIOCADDMULTI Enable a multicast address for the interface. A maximum of 64 multicast addresses may be enabled for any given interface.

SIOCDELMULTI Disable a previously set multicast address.

SIOCSPROMISC Toggle promiscuous mode.

SEE ALSO

arp(4P), **ec**(4S), **lo**(4)

NAME

inet – Internet protocol family

SYNOPSIS

options INET

#include <sys/types.h>
#include <netinet/in.h>

DESCRIPTION

The Internet protocol family implements a collection of protocols which are centered around the *Internet Protocol* (IP) and which share a common address format. The Internet family provides protocol support for the **SOCK_STREAM**, **SOCK_DGRAM**, and **SOCK_RAW** socket types.

PROTOCOLS

The Internet protocol family is comprised of the Internet Protocol (IP), the Address Resolution Protocol (ARP), the Internet Control Message Protocol (ICMP), the Transmission Control Protocol (TCP), and the User Datagram Protocol (UDP).

TCP is used to support the **SOCK_STREAM** abstraction while UDP is used to support the **SOCK_DGRAM** abstraction; see **tcp**(4P) and **udp**(4P). A raw interface to IP is available by creating an Internet socket of type **SOCK_RAW**; see **ip**(4P). ICMP is used by the kernel to handle and report errors in protocol processing. It is also accessible to user programs; see **icmp**(4P). ARP is used to translate 32-bit IP addresses into 48-bit Ethernet addresses; see **arp**(4P).

The 32-bit IP address is divided into network number and host number parts. It is frequency-encoded; the most-significant bit is zero in Class A addresses, in which the high-order 8 bits are the network number. Class B addresses have their high order two bits set to 10 and use the high-order 16 bits as the network number field. Class C addresses have a 24-bit network number part of which the high order three bits are 110. Sites with a cluster of local networks may chose to use a single network number for the cluster; this is done by using subnet addressing. The local (host) portion of the address is further subdivided into subnet number and host number parts. Within a subnet, each subnet appears to be an individual network; externally, the entire cluster appears to be a single, uniform network requiring only a single routing entry. Subnet addressing is enabled and examined by the following **ioctl**(2) commands on a datagram socket in the Internet domain; they have the same form as the **SIOCIFADDR** command (see **intro**(4N)).

SIOCSIFNETMASK Set interface network mask. The network mask defines the network part of the address; if it contains more of the address than the address type would indicate, then subnets are in use.

SIOCGIFNETMASK Get interface network mask.

ADDRESSING

IP addresses are four byte quantities, stored in network byte order (on Sun386i systems these are word and byte reversed).

Sockets in the Internet protocol family use the following addressing structure:

```
struct sockaddr_in {
        short   sin_family;
        u_short sin_port;
        struct  in_addr sin_addr;
        char    sin_zero[8];
};
```

Library routines are provided to manipulate structures of this form; see **intro**(3N).

The **sin_addr** field of the **sockaddr_in** structure specifies a local or remote IP address. Each network interface has its own unique IP address. The special value **INADDR_ANY** may be used in this field to effect "wildcard" matching. Given in a **bind**(2) call, this value leaves the local IP address of the socket unspecified, so that the socket will receive connections or messages directed at any of the valid IP addresses of the system. This can prove useful when a process neither knows nor cares what the local IP

address is or when a process wishes to receive requests using all of its network interfaces. The **sockaddr_in** structure given in the **bind**(2) call must specify an **in_addr** value of either **IPADDR_ANY** or one of the system's valid IP addresses. Requests to bind any other address will elicit the error EADDRNOTAVAIL. When a **connect**(2) call is made for a socket that has a wildcard local address, the system sets the **sin_addr** field of the socket to the IP address of the network interface that the packets for that connection are routed via.

The **sin_port** field of the **sockaddr_in** structure specifies a port number used by TCP or UDP. The local port address specified in a **bind**(2) call is restricted to be greater than **IPPORT_RESERVED** (defined in **<netinet/in.h>**) unless the creating process is running as the super-user, providing a space of protected port numbers. In addition, the local port address must not be in use by any socket of same address family and type. Requests to bind sockets to port numbers being used by other sockets return the error EADDRINUSE. If the local port address is specified as 0, then the system picks a unique port address greater than **IPPORT_RESERVED**. A unique local port address is also picked when a socket which is not bound is used in a **connect**(2) or **sendto** (see **send**(2)) call. This allows programs which do not care which local port number is used to set up TCP connections by simply calling **socket**(2) and then **connect**(2), and to send UDP datagrams with a **socket**(2) call followed by a **sendto**(2) call.

Although this implementation restricts sockets to unique local port numbers, TCP allows multiple simultaneous connections involving the same local port number so long as the remote IP addresses or port numbers are different for each connection. Programs may explicitly override the socket restriction by setting the **SO_REUSEADDR** socket option with **setsockopt** (see **getsockopt**(2)).

SEE ALSO

bind(2), **connect**(2), **getsockopt**(2), **ioctl**(2), **sendto**(2), **socket**(2), **byteorder**(3N), **gethostent**(3N), **getnetent**(3N), **getprotoent**(3N), **getservent**(3N), **inet**(3N), **intro**(3N), **arp**(4P), **icmp**(4P), **intro**(4N), **ip**(4P) **tcp**(4P), **udp**(4P),

Network Information Center, *DDN Protocol Handbook* (3 vols.), Network Information Center, SRI International, Menlo Park, Calif., 1985.
A 4.2BSD Interprocess Communication Primer

CAVEAT

The Internet protocol support is subject to change as the Internet protocols develop. Users should not depend on details of the current implementation, but rather the services exported.

NAME

ip – Internet Protocol

SYNOPSIS

#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_RAW, proto);

DESCRIPTION

IP is the internetwork datagram delivery protocol that is central to the Internet protocol family. Programs may use IP through higher-level protocols such as the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP), or may interface directly using a "raw socket." See **tcp**(4P) and **udp**(4P). The protocol options defined in the IP specification may be set in outgoing datagrams.

Raw IP sockets are connectionless and are normally used with the **sendto** and **recvfrom** calls, (see **send**(2) and **recv**(2)) although the **connect**(2) call may also be used to fix the destination for future datagrams (in which case the **read**(2V) or **recv**(2) and **write**(2V) or **send**(2) calls may be used). If **proto** is zero, the default protocol, **IPPROTO_RAW**, is used. If **proto** is non-zero, that protocol number will be set in outgoing datagrams and will be used to filter incoming datagrams. An IP header will be generated and prepended to each outgoing datagram; Received datagrams are returned with the IP header and options intact.

A single socket option, **IP_OPTIONS**, is supported at the IP level. This socket option may be used to set IP options to be included in each outgoing datagram. IP options to be sent are set with **setsockopt** (see **getsockopt**(2)). The **getsockopt**(2) call returns the IP options set in the last **setsockopt** call. IP options on received datagrams are visible to user programs only using raw IP sockets. The format of IP options given in **setsockopt** matches those defined in the IP specification with one exception: the list of addresses for the source routing options must include the first-hop gateway at the beginning of the list of gateways. The first-hop gateway address will be extracted from the option list and the size adjusted accordingly before use. IP options may be used with any socket type in the Internet family.

At the socket level, the socket option **SO_DONTROUTE** may be applied. This option forces datagrams being sent to bypass the routing step in output. Normally, IP selects a network interface to send the datagram via, and possibly an intermediate gateway, based on an entry in the routing table. See **routing**(4N). When **SO_DONTROUTE** is set, the datagram will be sent via the interface whose network number or full IP address matches the destination address. If no interface matches, the error ENETUNRCH will be returned.

Datagrams flow through the IP layer in two directions: from the network **ip** to user processes and from user processes *down* to the network. Using this orientation, IP is layered *above* the network interface drivers and *below* the transport protocols such as UDP and TCP. The Internet Control Message Protocol (ICMP) is logically a part of IP. See **icmp**(4P).

IP provides for a checksum of the header part, but not the data part of the datagram. The checksum value is computed and set in the process of sending datagrams and checked when receiving datagrams. IP header checksumming may be disabled for debugging purposes by patching the kernel variable **ipcksum** to have the value zero.

IP options in received datagrams are processed in the IP layer according to the protocol specification. Currently recognized IP options include: security, loose source and record route (LSRR), strict source and record route (SSRR), record route, stream identifier, and internet timestamp.

The IP layer will normally forward received datagrams that are not addressed to it. Forwarding is under the control of the kernel variable *ipforwarding*: if *ipforwarding* is zero, IP datagrams will not be forwarded; if *ipforwarding* is one, IP datagrams will be forwarded. *ipforwarding* is usually set to one only in machines with more than one network interface (internetwork routers). This kernel variable can be patched to enable or disable forwarding.

The IP layer will send an ICMP message back to the source host in many cases when it receives a datagram that can not be handled. A "time exceeded" ICMP message will be sent if the "time to live" field in the IP header drops to zero in the process of forwarding a datagram. A "destination unreachable" message will be sent if a datagram can not be forwarded because there is no route to the final destination, or if it can not be fragmented. If the datagram is addressed to the local host but is destined for a protocol that is not supported or a port that is not in use, a destination unreachable message will also be sent. The IP layer may send an ICMP "source quench" message if it is receiving datagrams too quickly. ICMP messages are only sent for the first fragment of a fragmented datagram and are never returned in response to errors in other ICMP messages.

The IP layer supports fragmentation and reassembly. Datagrams are fragmented on output if the datagram is larger than the maximum transmission unit (MTU) of the network interface. Fragments of received datagrams are dropped from the reassembly queues if the complete datagram is not reconstructed within a short time period.

Errors in sending discovered at the network interface driver layer are passed by IP back up to the user process.

ERRORS

A socket operation may fail with one of the following errors returned:

EACCESS	when specifying an IP broadcast destination address if the caller is not the superuser;
EISCONN	when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected;
EMSGSIZE	when sending datagram that is too large for an interface, but is not allowed be fragmented (such as broadcasts);
ENETUNREACH	when trying to establish a connection or send a datagram, if there is no matching entry in the routing table, or if an ICMP "destination unreachable" message is received.
ENOTCONN	when trying to send a datagram, but no destination address is specified, and the socket hasn't been connected;
ENOBUFS	when the system runs out of memory for fragmentation buffers or other internal data structure;
EADDRNOTAVAIL	when an attempt is made to create a socket with a local address that matches no network interface, or when specifying an IP broadcast destination address and the network interface does not support broadcast;

The following errors may occur when setting or getting IP options:

EINVAL	An unknown socket option name was given.
EINVAL	The IP option field was improperly formed; an option field was shorter than the minimum value or longer than the option buffer provided.

SEE ALSO

connect(2), **getsockopt**(2), **read**(2V), **recv**(2), **send**(2), **write**(2V), **icmp**(4P), **inet**(4F) **routing**(4N), **tcp**(4P), **udp**(4P),

Postel, Jon, "*Internet Protocol - DARPA Internet Program Protocol Specification*," RFC 791, Network Information Center, SRI International, Menlo Park, Calif., September 1981. (Sun 800-1063-01)

BUGS

Raw sockets should receive ICMP error packets relating to the protocol; currently such packets are simply discarded.

Users of higher-level protocols such as TCP and UDP should be able to see received IP options.

NAME

ipi – IPI driver

CONFIG — SUN-3 and SUN-4 SYSTEMS

controller ipi *cpu-type* **at nexus ?**

DESCRIPTION

The controller line above declares the psuedo-bus IPI, providing a connection between IPI device drivers and IPI channel drivers (also known as host adaptor drivers). Declaring the bus pulls in this middle layer of the IPI drivers, so the IPI bus declaration should be omitted if there are no IPI drivers. See **id**(4S) and **is**(4S). The cpu-type number should be the same as that used in the **obio** declaration.

FILES

This driver supports only an interface to other drivers, and therefore does not have any special file access to user programs.

SEE ALSO

id(4S), **is**(4S)

DIAGNOSTICS

ipi_alloc *csf*: **invalid IPI address**
A device driver called **ipi_alloc()** for a device on a non-existent channel or for an illegal slave address.

ipi_async: ipi *csf* **slave number out of range**
A channel driver called **ipi_async()** for with a response packet containting an illegal slave address.

ipi_setup *csf*: **invalid IPI address**
An IPI device driver used an incorrect IPI address. The channel, slave, or facility number was out of range. This could be a configuration problem.

ipi *csf*: **channel not configured**
This message is printed when a channel driver makes an invalid call to the IPI driver.

ipi_async: IPI channel *c* **not configured**
The channel driver made an invalid call to **ipi_async()**.

ipi_async: ipi Iccssff slave number out of range
The channel driver made an invalid call to **ipi_async()**.

ipi_channel: No memory to add channel *c*
Dynamic memory allocation failed.

ipi_channel: no memory to alloc channel *c*
Dynamic memory allocation failed.

ipi_channel: channel *c* **already defined**
A channel driver tried to add a channel which was previously assigned to a channel driver. This could be caused by a configuration problem.

ipi_free_chan: channel *c* **not allocated**
A channel driver attempted to delete a channel which had never been added.

ipi channel *c* **deleted**
A channel driver deleted the channel because the channel device failed to respond.

ipi: assign_refnum: q *addr* **already has refnum** *r*
A command reference number is being assigned to a command that already had a reference number. This indicates a driver problem.

free_refnum: q *addr* **refnum** *r* **lookup** *l-addr*
The driver was freeing an IPI request at address *addr*, and found that the reference number it used, *r*, was assigned to another request at address *l-addr*.

ipi_lookup: found wrong command for refnum
ipi_lookup: looking for *r*, found *fr* q *addr*
The driver was looking for a command with a particular reference number, *r*, but the lookup table found one with a different reference number, *fr*. These messages are always received together.

ipi *csf*: missing interrupt. refnum *r*
The driver detected that a command took longer than the device driver expected it to take. This message will be followed by other messages from the device driver.

NAME

is – IPI channel driver for Sun IPI string controllers

CONFIG — SUN-3, SUN-4 SYSTEMS

```
controller      isc0    at vme32d32 ? csr 0x01080000 priority 2 vector isintr 0x4c
channel         is0     at isc0 ipi_addr 0x00000 # channel 0 slave 0
controller      isc1    at vme32d32 ? csr 0x01080400 priority 2 vector isintr 0x4d
channel         is1     at isc1 ipi_addr 0x00100 # channel 0 slave 1
controller      isc2    at vme32d32 ? csr 0x01080800 priority 2 vector isintr 0x4e
channel         is2     at isc2 ipi_addr 0x00200 # channel 0 slave 2
controller      isc3    at vme32d32 ? csr 0x01080c00 priority 2 vector isintr 0x4f
channel         is3     at isc3 ipi_addr 0x00300 # channel 0 slave 3
```

DESCRIPTION

The four **controller** and **channel** lines given in the synopsis section above correspond to the first, second, third, and fourth ISP-80 IPI disk controller in a Sun system. These controllers are treated as integrated channel/disk-controller cards, in that there are two drivers that manage them. This driver, the **is** driver, presents a generic IPI-3 interface to the **id** driver, which handles the controller as if it was a channel-attached IPI-3 controller. See **id**(4S).

The **csr** value gives the VME address of the controller. The **ipi_addr** field gives the 3-byte channel, slave, and facility address supported by the controller, the facility address portion is ignored and should be specified as zero.

SEE ALSO

id(4S)

DIAGNOSTICS

isc*n*: channel reset timed out
The probe routine, called during system initialization, determined that the controller reset, initiated by the boot, had not completed in the allotted time. The controller cannot be used.

isc*n*: bad IPI address *addr*
The specified address contained a slave address larger than 7.

isc*n*: interrupt vector not specified in config
The configuration did not specify an interrupt vector for the controller.

is*n*: refnum lookup on success failed. refnum *refnum*
A completion interrupt occurred for a command which was unknown to the controller. The interrupt is ignored.

is*n*: response too short. len *l* **min** *m* **response** *r*
The response packet presented in an interrupt was too short to contain a valid header. The response is printed, but otherwise the interrupt is ingored.

is*n*: response too long. max *m* **len** *l* **truncating**
The response packet is longer than the maximum, *m*, for this controller,

is*n*: is_reset_slave: resetting slave
The driver is resetting the controller under direction from the **id** disk driver. This occurs during recovery from timed out requests, also known as missing interrupts.

is*n*: is_reset_slave: slave reset not enabled
The **id** driver attempted to reset the controller to recovery from an error, but the driver has been set to disable such resets.

is*n*: ctlr reset timed out
The reset of the controller took longer than expected and has presumably failed.

isc*n*: response reg *r* **not same as packet refnum** *ref*
After reading a response packet, the response register was read and did not match the command

reference number in the response packet. This could indicate a problem with the driver or the controller.

isc*n*: error status *s* response *r*
The controller presented error status code *s* and a response register value of *r*. These will be interpreted by messages that follow.

isc*n*: ctlr fault *f* - bus error on cmd refnum *r*
The controller reported a bus error while fetching the command.

isc*n*: ctlr fault *f* - timeout on cmd refnum *r*
The controller reported a VME timeout while fetching the command.

isc*n*: ctlr fault *f* - invalid command reg write on cmd refnum *r*
The controller indicates that the command register was written after the controller has panicked.

isc*n*: ctlr fault *f* - unknown fault code on cmd refnum *r*
The controller has presented an unknown fault code in it's status.

NAME

lofs – loopback virtual file system

CONFIG

options LOFS

SYNOPSIS

#include <sys/mount.h>
mount(MOUNT_LOFS, virtual, flags, dir);

DESCRIPTION

The loopback file system device allows new, virtual file systems to be created, which provide access to existing files using alternate pathnames. Once the virtual file system is created, other file systems can be mounted within it without affecting the original file system. File systems that are subsequently mounted onto the original file system, however, *are* visible to the virtual file system, unless or until the corresponding mount point in the virtual file system is covered by a file system mounted there.

virtual is the mount point for the virtual file system. *dir* is the pathname of the existing file system. *flags* is either 0 or **M_RDONLY**. The **M_RDONLY** flag forces all accesses in the new name space to be read-only; without it, accesses are the same as for the underlying file system. All other **mount**(2) flags are preserved from the underlying file systems.

A loopback mount of '/' onto **/tmp/newroot** allows the entire file system hierarchy to appear as if it were duplicated under **/tmp/newroot**, including any file systems mounted from remote NFS servers. All files would then be accessible either from a pathname relative to '/', or from a pathname relative to **/tmp/newroot** until such time as a file system is mounted in **/tmp/newroot**, or any of its subdirectories.

Loopback mounts of '/' can be performed in conjunction with the **chroot**(2) system call, to provide a complete virtual file system to a process or family of processes.

Recursive traversal of loopback mount points is not allowed; after the loopback mount of **/tmp/newroot**, the file **/tmp/newroot/tmp/newroot** does not contain yet another file system hierarchy; rather, it appears just as **/tmp/newroot** did before the loopback mount was performed (say, as an empty directory).

The standard RC files perform first **4.2** mounts, then **nfs** mounts, during booting. On Sun386*i* systems, **lo** (loopback) mounts are performed just after **4.2** mounts. **/etc/fstab** files depending on alternate mount orders at boot time will fail to work as expected. Manual modification of **/etc/rc.local** will be needed to make such mount orders work.

WARNINGS

Loopback mounts must be used with care; the potential for confusing users and applications is enormous. A loopback mount entry in **/etc/fstab** must be placed after the mount points of both directories it depends on. This is most easily accomplished by making the loopback mount entry the last in **/etc/fstab**, though see **mount**(8) for further warnings.

SEE ALSO

chroot(2), **mount**(2), **fstab**(5), **mount**(8)

BUGS

Because only directories can be mounted or mounted on, the structure of a virtual file system can only be modified at directories.

NAME

mcp, alm – Sun MCP Multiprotocol Communications Processor/ALM-2 Asynchronous Line Multiplexer

CONFIG — SUN-3, SUN-3x, AND SUN-4 SYSTEMS

MCP

```
device mcp0 at vme32d32 ? csr 0x1000000 flags 0x1ffff priority 4 vector mcpintr 0x8b
device mcp1 at vme32d32 ? csr 0x1010000 flags 0x1ffff priority 4 vector mcpintr 0x8a
device mcp2 at vme32d32 ? csr 0x1020000 flags 0x1ffff priority 4 vector mcpintr 0x89
device mcp3 at vme32d32 ? csr 0x1030000 flags 0x1ffff priority 4 vector mcpintr 0x88
device mcp4 at vme32d32 ? csr 0x2000000 flags 0x1ffff priority 4 vector mcpintr 0x8b
device mcp5 at vme32d32 ? csr 0x2010000 flags 0x1ffff priority 4 vector mcpintr 0x8a
device mcp6 at vme32d32 ? csr 0x2020000 flags 0x1ffff priority 4 vector mcpintr 0x89
device mcp7 at vme32d32 ? csr 0x2030000 flags 0x1ffff priority 4 vector mcpintr 0x88
```

ALM-2

```
pseudo-device mcpa128
```

SYNOPSIS

```
#include <fcntl.h>
#include <sys/termios.h>
open("/dev/ttyxy", mode);
open("/dev/ttyd n", mode);
open("/dev/cua n", mode);
```

DESCRIPTION (MCP)

The Sun MCP (Multiprotocol Communications Processor) supports up to four synchronous serial lines in conjunction with SunLink™ Multiple Communication Protocol products.

DESCRIPTION (ALM-2)

The Sun ALM-2 Asynchronous Line Multiplexer provides 16 asynchronous serial communication lines with modem control and one Centronics-compatible parallel printer port.

Each port supports those **termio**(4) device control functions specified by flags in the **c_cflag** word of the **termios** structure and by the **IGNBRK**, **IGNPAR**, **PARMRK**, or **INPCK** flags in the **c_iflag** word of the **termios** structure are performed by the **mcp** driver. All other **termio**(4) functions must be performed by STREAMS modules pushed atop the driver; when a device is opened, the **ldterm**(4M) and **ttcompat**(4M) STREAMS modules are automatically pushed on top of the stream, providing the standard **termio**(4) interface.

Bit *i* of **flags** may be specified to say that a line is not properly connected, and that the line *i* should be treated as hard-wired with carrier always present. Thus specifying **flags 0x0004** in the specification of **mcp0** would treat line **/dev/ttyh2** in this way.

Minor device numbers in the range 0 – 127 correspond directly to the normal tty lines and are named **/dev/tty***XY*, where *X* represents the physical board as one of the characters **h**, **i**, **j**, **k**, **l**, **m**, **n**, or **o**, and *Y* is the line number on the board as a single hexadecimal digit. Thus the first line on the first board is **/dev/ttyh0**, and the sixteenth line on the third board is **/dev/ttyjf**.

To allow a single tty line to be connected to a modem and used for both incoming and outgoing calls, a special feature, controlled by the minor device number, has been added. Minor device numbers in the range 128 – 255 correspond to the same physical lines as those above (that is, the same line as the minor device number minus 128).

A dial-in line has a minor device in the range 0 – 127 and is conventionally renamed **/dev/ttyd***n*, where *n* is a number indicating which dial-in line it is (so that **/dev/ttyd0** is the first dial-in line), and the dial-out line corresponding to that dial-in line has a minor device number 128 greater than the minor device number of the dial-in line and is conventionally named **/dev/cua***n*, where *n* is the number of the dial-in line.

The **/dev/cua***n* lines are special in that they can be opened even when there is no carrier on the line. Once a **/dev/cua***n* line is opened, the corresponding tty line cannot be opened until the **/dev/cua***n* line is closed; a blocking open will wait until the **/dev/cua***n* line is closed (which will drop Data Terminal Ready, after which Carrier Detect will usually drop as well) and carrier is detected again, and a non-blocking open will return an error. Also, if the **/dev/ttyd***n* line has been opened successfully (usually only when carrier is recognized on the modem) the corresponding **/dev/cua***n* line cannot be opened. This allows a modem to be attached to e.g. **/dev/ttyd0** (renamed from **/dev/ttyh0**) and used for dialin (by enabling the line for login in **/etc/ttytab**) and also used for dialout (by **tip**(1C) or **uucp**(1C)) as **/dev/cua0** when no one is logged in on the line. Note: the bit in the **flags** word in the configuration file (see above) must be zero for this line, which enables hardware carrier detection.

IOCTLS

The standard set of **termio ioctl**( ) calls are supported by the ALM-2.

If the **CRTSCTS** flag in the **c_cflag** is set, output will be generated only if CTS is high; if CTS is low, output will be frozen. If the **CRTSCTS** flag is clear, the state of CTS has no effect. Breaks can be generated by the **TCSBRK**, **TIOCSBRK**, and **TIOCCBRK ioctl**( ) calls. The modem control lines **TIOCM_CAR**, **TIOCM_CTS**, **TIOCM_RTS**, and **TIOCM_DTR** are provided.

The input and output line speeds may be set to any of the speeds supported by **termio**. The speeds cannot be set independently; when the output speed is set, the input speed is set to the same speed.

ERRORS

An **open**( ) on a **/dev/tty*** or a **/dev/cu*** device will fail if:

ENXIO	The unit being opened does not exist.
EBUSY	The dial-out device is being opened and the dial-in device is already open, or the dial-in device is being opened with a no-delay open and the dial-out device is already open.
EBUSY	The unit has been marked as exclusive-use by another process with a **TIOCEXCL ioctl**( ) call.
EINTR	The open was interrupted by the delivery of a signal.

DESCRIPTION (PARALLEL PORT)

The parallel port is Centronics-compatible and is suitable for most common parallel printers. Devices attached to this interface are normally handled by the line printer spooling system, and should not be accessed directly by the user.

The printer devices reside on a separate major device number from the serial devices. Minor device numbers in the range 0 – 7 access the printer, and the recommended naming is **/dev/mcpp[0-7]**.

IOCTLS

Various control flags and status bits may be fetched and set on an MCP printer port. The following flags and status bits are supported; they are defined in **sundev/mcpcmd.h**:

MCPRIGNSLCT	0x02	set if interface ignoring SLCT– on open
MCPRDIAG	0x04	set if printer port is in self-test mode
MCPRVMEINT	0x08	set if VME bus interrupts enabled
MCPRINTPE	0x10	print message when out of paper
MCPRINTSLCT	0x20	print message when printer offline
MCPRPE	0x40	set if device ready, cleared if device out of paper
MCPRSLCT	0x80	set if device online (Centronics SLCT asserted)

The flags **MCPRINTSLCT, MCPRINTPE,** and **MCPRDIAG** may be changed; the other bits are status bits and may not be changed.

The **ioctl**( ) calls supported by MCP printer ports are listed below.

MCPIOGPR	The argument is a pointer to an **unsigned char**. The printer flags and status bits are stored in the **unsigned char** pointed to by the argument.

MCPIOSPR The argument is a pointer to an **unsigned char**. The printer flags are set from the **unsigned char** pointed to by the argument.

ERRORS

Normally, the interface only reports the status of the device when attempting an **open**(2V) call. An **open**() on a **/dev/mcpp*** device will fail if:

ENXIO The unit being opened does not exist.

EIO The device is offline or out of paper.

Bit 17 of the configuration **flags** may be specified to say that the interface should ignore Centronics SLCT– and RDY/PE– when attempting to open the device, but this is normally useful only for configuration and troubleshooting: if the SLCT– and RDY lines are not asserted during an actual data transfer (as with a **write**(2V) call), no data is transferred.

FILES

/dev/mcpp[0-7]	parallel printer port
/dev/tty[h-o][0-9a-f]	hardwired tty lines
/dev/ttyd[0-9a-f]	dialin tty lines
/dev/cua[0-9a-f]	dialout tty lines

SEE ALSO

tip(1C), **uucp**(1C), **mti**(4S), **termio**(4), **ldterm**(4M), **ttcompat**(4M), **zs**(4S)

DIAGNOSTICS

Most of these diagnostics "should never happen;" their occurrence usually indicates problems elsewhere in the system as well.

mcpa*n*: silo overflow.
More than *n* characters (*n* very large) have been received by the **mcp** hardware without being read by the software.

port *n* supports RS449 interface**
Probably an incorrect jumper configuration. Consult the hardware manual.

mcp port *n* receive buffer error
The **mcp** encountered an error concerning the synchronous receive buffer.

Printer on mcpp*n* is out of paper
Printer on mcpp*n* paper ok
Printer on mcpp*n* is offline
Printer on mcpp*n* online
Assorted printer diagnostics, if enabled as discussed above.

BUGS

Note: pin 4 is used for hardware flow control on ALM–2 ports 0 through 3. These two pins should *not* be tied together on the ALM end.

NAME

mouse – Sun mouse

CONFIG

None; included in standard system.

DESCRIPTION

The **mouse** indirect device provides access to the Sun Workstation mouse. When opened, it redirects operations to the standard mouse device for the workstation (attached either to a CPU serial or parallel port), and pushes the **ms**(4M) and **ttcompat**(4M) STREAMS modules on top of that device.

FILES

/dev/mouse

SEE ALSO

ms(4M), **ttcompat**(4M), **win**(4S), **zs**(4S)

NAME

ms – Sun mouse STREAMS module

CONFIG

pseudo-device*ms***n**

SYNOPSIS

```
#include <sys/stream.h>
#include <sys/stropt.h>
#include <sundev/vuid_event.h>
#include <sundev/msio.h>
ioctl(fd, I_PUSH, "ms");
```

DESCRIPTION

The **ms** STREAMS module processes byte streams generated by mice attached to a CPU serial or parallel port. When this module is pushed onto a stream, it sends a **TCSETSF** ioctl downstream, setting the baud rate to 1200 baud and the character size to 8 bits, and enabling the receiver. All other flag words are cleared. It assumes only that the **termios**(4) functions provided by the **zs**(4S) driver are supported; no other functions need be supported.

The mouse is expected to generate a stream of bytes encoding mouse motions and changes in the state of the buttons.

Each mouse sample in the byte stream consists of three bytes: the first byte gives the button state with value 0x87|~*but*, where *but* is the low three bits giving the mouse buttons, where a 0 (zero) bit means that a button is pressed, and a 1 (one) bit means a button is not pressed. Thus if the left button is down the value of this sample is 0x83, while if the right button is down the byte is 0x86.

The next two bytes of each sample give the *x* and *y* deltas of this sample as signed bytes. The mouse uses a lower-left coordinate system, so moves to the right on the screen yield positive *x* values and moves down the screen yield negative *y* values.

The beginning of a sample is identifiable because the delta's are constrained to not have values in the range 0x80-0x87.

A stream with **ms** pushed onto it can be used as a device that emits *firm_events* as specified by the protocol of a *Virtual User Input Device*. It understands **VUIDSFORMAT, VUIDGFORMAT, VUIDSADDR** and **VUIDGADDR** ioctls (see reference below).

IOCTLS

ms responds to the following *ioctl*s, as defined in **<sundev/msio.h>** and **<sundev/vuid_event.h>**. All other *ioctl*s are passed downstream. As **ms** sets the parameters of the serial port when it is opened, no **termios**(4) *ioctl*s should be performed on a stream with **ms** on it, as **ms** expects the device parameters to remain as it set them.

The **MSIOGETPARMS** and **MSIOSETPARMS** calls use a structure of type **Ms_parms**, which is a structure defined in **<sundev/msio.h>**:

```
typedef struct {
int         jitter_thresh;
int         speed_law;
int         speed_limit;
}           Ms_parms;
```

jitter_thresh is the "jitter threshold" of the mouse. Motions of fewer than *jitter_thresh* units along both axes that occur in less than 1/12 second are treated as "jitter" and ignored. Thus, if the mouse moves fewer than *jitter_thresh units* and then moves back to its original position in less than 1/12 of a second, the motion is considered to be "noise" and ignored. If it moves fewer than *jitter_thresh* units and continues to move so that it has not returned to its original position after 1/12 of a second, the motion is considered to be real and is reported.

speed_law indicates whether extremely large motions are to be ignored. If it is 1, a "speed limit" is applied to mouse motions; motions along either axis of more than *speed_limit* units are discarded.

Note: these parameters are global; if they are set for any mouse on a workstation, they apply to any other mice attached to that workstation as well.

VUIDSFORMAT
VUIDGFORMAT
VUIDSADDR
VUIDGADDR These are standard *Virtual User Input Device ioctl*s. See *SunView 1 System Programmer's Guide* for a description of their operation.

MSIOGETPARMS The argument is a pointer to a **Ms_parms**. The current mouse parameters are stored in that structure.

MSIOSETPARMS The argument is a pointer to a **ms_parms**. The current mouse parameters are set from the values in that structure.

SEE ALSO

mouse(4S), **termios**(4), **win**(4S), **zs**(4S)

SunView 1 System Programmer's Guide

NAME

mti – Systech MTI-800/1600 multi-terminal interface

CONFIG — SUN-3 SYSTEM

device mti0 at vme16d16 ? csr 0x620 flags 0xffff priority 4 vector mtiintr 0x88
device mti1 at vme16d16 ? csr 0x640 flags 0xffff priority 4 vector mtiintr 0x89
device mti2 at vme16d16 ? csr 0x660 flags 0xffff priority 4 vector mtiintr 0x8a
device mti3 at vme16d16 ? csr 0x680 flags 0xffff priority 4 vector mtiintr 0x8b

CONFIG — SUN-2 SYSTEM

device mti0 at mbio ? csr 0x620 flags 0xffff priority 4
device mti1 at mbio ? csr 0x640 flags 0xffff priority 4
device mti2 at mbio ? csr 0x660 flags 0xffff priority 4
device mti3 at mbio ? csr 0x680 flags 0xffff priority 4
device mti0 at vme16 ? csr 0x620 flags 0xffff priority 4 vector mtiintr 0x88
device mti1 at vme16 ? csr 0x640 flags 0xffff priority 4 vector mtiintr 0x89
device mti2 at vme16 ? csr 0x660 flags 0xffff priority 4 vector mtiintr 0x8a
device mti3 at vme16 ? csr 0x680 flags 0xffff priority 4 vector mtiintr 0x8b

SYNOPSIS

#include <fcntl.h>
#include <sys/termios.h>
open("/dev/tty*xy*", mode);
open("/dev/ttyd*n*", mode);
open("/dev/cua*n*", mode);

DESCRIPTION

The Systech MTI card provides 8 (MTI-800) or 16 (MTI-1600) serial communication lines with modem control. Each port supports those **termio**(4) device control functions specified by flags in the **c_cflag** word of the **termios** structure and by the **IGNBRK**, **IGNPAR**, **PARMRK**, or **INPCK** flags in the **c_iflag** word of the **termios** structure are performed by the **mti** driver. All other **termio**(4) functions must be performed by STREAMS modules pushed atop the driver; when a device is opened, the **ldterm**(4M) and **ttcompat**(4M) STREAMS modules are automatically pushed on top of the stream, providing the standard **termio**(4) interface.

Bit *i* of **flags** may be specified to say that a line is not properly connected, and that the line *i* should be treated as hard-wired with carrier always present. Thus specifying **flags 0x0004** in the specification of **mti0** would treat line **/dev/tty02** in this way.

Minor device numbers in the range 0 – 63 correspond directly to the normal tty lines and are named **/dev/tty***XY*, where *X* is the physical board number (0 – 3), and *Y* is the line number on the board as a single hexadecimal digit. (Thus the first line on the first board is **/dev/tty00**, and the sixteenth line on the third board is **/dev/tty2f**.)

To allow a single tty line to be connected to a modem and used for both incoming and outgoing calls, a special feature, controlled by the minor device number, has been added. Minor device numbers in the range 128 – 191 correspond to the same physical lines as those above (that is, the same line as the minor device number minus 128).

A dial-in line has a minor device in the range 0 – 63 and is conventionally renamed **/dev/ttyd***n*, where *n* is a number indicating which dial-in line it is (so that **/dev/ttyd0** is the first dial-in line), and the dial-out line corresponding to that dial-in line has a minor device number 128 greater than the minor device number of the dial-in line and is conventionally named **/dev/cua***n*, where *n* is the number of the dial-in line.

The **/dev/cua***n* lines are special in that they can be opened even when there is no carrier on the line. Once a **/dev/cua***n* line is opened, the corresponding tty line can not be opened until the **/dev/cua***n* line is closed; a blocking open will wait until the **/dev/cua***n* line is closed (which will drop Data Terminal Ready, after which Carrier Detect will usually drop as well) and carrier is detected again, and a non-blocking open will return an error. Also, if the **/dev/ttyd***n* line has been opened successfully (usually only when carrier is

recognized on the modem) the corresponding **/dev/cua*n*** line can not be opened. This allows a modem to be attached to e.g. **/dev/ttyd0** (renamed from **/dev/tty00**) and used for dialin (by enabling the line for login in **/etc/ttytab**) and also used for dialout (by **tip**(1C) or **uucp**(1C)) as **/dev/cua0** when no one is logged in on the line. Note: the bit in the **flags** word in the configuration file (see above) must be zero for this line, which enables hardware carrier detection.

WIRING

The Systech requires the CTS modem control signal to operate. If the device does not supply CTS then RTS should be jumpered to CTS at the distribution panel (short pins 4 to 5). Also, the CD (carrier detect) line does not work properly. When connecting a modem, the modem's CD line should be wired to DSR, which the software will treat as carrier detect.

IOCTLS

The standard set of **termio ioctl**( ) calls are supported by **mti**.

The state of the **CRTSCTS** flag in the **c_cflag** word has no effect; no output will be generated unless CTS is high. Breaks can be generated by the **TCSBRK**, **TIOCSBRK**, and **TIOCCBRK ioctl**( ) calls. The modem control lines **TIOCM_CAR**, **TIOCM_CTS**, **TIOCM_RTS**, and **TIOCM_DTR** are provided; however, as described above, the DSR line is treated as CD and the CD line is ignored.

The input and output line speeds may be set to any of the speeds supported by **termio**. The speeds cannot be set independently; when the output speed is set, the input speed is set to the same speed. The baud rates **B200** and **B38400** are not supported by the hardware; **B200** selects 2000 baud, and **B38400** selects 7200 baud.

ERRORS

An **open**( ) will fail if:

ENXIO	The unit being opened does not exist.
EBUSY	The dial-out device is being opened and the dial-in device is already open, or the dial-in device is being opened with a no-delay open and the dial-out device is already open.
EBUSY	The unit has been marked as exclusive-use by another process with a **TIOCEXCL ioctl**( ) call.
EINTR	The open was interrupted by the delivery of a signal.

FILES

/dev/tty[0-3][0-9a-f]	hardwired tty lines
/dev/ttyd[0-9a-f]	dialin tty lines
/dev/cua[0-9a-f]	dialout tty lines

SEE ALSO

tip(1C), **uucp**(1C), **mcp**(4S), **termio**(4), **ldterm**(4M), **ttcompat**(4M), **zs**(4S)

DIAGNOSTICS

Most of these diagnostics "should never happen" and their occurrence usually indicates problems elsewhere in the system.

mti*n*, *n*: silo overflow.
More than 512 characters have been received by the mti hardware without being read by the software. Extremely unlikely to occur.

mti*n*: read error code <*n*>. Probable hardware fault
The **mti** returned the indicated error code. See the MTI manual.

mti*n*: DMA output error.
The **mti** encountered an error while trying to do DMA output.

mti*n*: impossible response *n*.
The **mti** returned an error it could not understand.

NAME

mtio – general magnetic tape interface

SYNOPSIS

#include <sys/ioctl.h>
#include <sys/mtio.h>

DESCRIPTION

Both 1/2" and 1/4" magnetic tape drives share the same general interface regardless of the hardware involved. The remainder of this section discusses the common features of that interface.

The "cooked" magnetic tape device files read and write magnetic tape in 2048 byte blocks (the 2048 is actually **BLKDEV_IOSIZE** in **<sys/param.h>**). The name of such a device file might be **/dev/mt0**. The final component of the device name is the type of device the file refers to, and the unit number of that device.

These files are rewound when closed, except for "no-rewind" versions. The names of no-rewind device files use the letter **n** as the beginning of the final component. The no-rewind version of **/dev/mt0** would be **/dev/nmt0**. When a 1/2" tape file, opened for writing or just written, is closed, two tape marks are written. If the tape is not to be rewound, it is positioned with the head between the two tapemarks.

The files discussed above are useful for making taped files consistent with ordinary files. This interface requires that all blocks be 2048 bytes long, and does not permit special operations (such as spacing the tape forward or backward) to be performed. The "raw" interface is appropriate when reading or writing long records or using foreign tapes. Raw device files are indicated by the letter **r** before the device type in the device name: the raw version of **/dev/mt0** would be **/dev/rmt0**, and the raw version of **/dev/nmt0** would be **/dev/nrmt0**.

read(2V) or **write**(2V) calls read or write the next record on the tape. When the call is to write, the record has the same length as the given buffer. During a read call, the record size is passed back as the number of bytes read, provided it is no greater than the buffer size. In raw tape I/O, seeks are ignored (except **st**). When a tape mark is read, a zero byte count is returned; another read will fetch the first record of the next tape file. Two successive reads returning zero byte counts indicate the end of recorded media.

1/2" Reel Tape

Data bytes are recorded in parallel onto the 9–track tape. The number of bytes in a physical record varies between 1 to 65535 bytes. Files end with one file mark except for the last, which signals the end of recorded media with two file marks. Care should be taken when overwriting records; the erase head is just forward of the write head and any following records will also be erased.

The recording formats available (check specific tape drive) are 800 BPI, 1600 BPI, and 6250 BPI, and data compression. Actual storage capacity is a function of the recording format and the length of the tape reel. For example, using a 2400 foot tape, 20 MB can be stored using 800 BPI, 40 MB using 1600 BPI, 140 MB using 6250 BPI, or up to 700 MB using data compression.

1/4" Cartridge Tape

Data is recorded serially onto 1/4" cartridge tape. The number of bytes per record is determined by the physical record size of the device. The I/O request size must be a multiple of the physical record size of the device. For QIC–11, QIC–24, and QIC–150 tape drives the block size is 512 bytes.

The records are recorded on tracks in a serpentine motion. As one track is completed, the drive switches to the next and begins writing in the opposite direction, eliminating the wasted motion of rewinding. Each file, including the last, ends with one file mark.

Files may be written only at the beginning of the tape or after the last written file. This prevents corrupting data by overwriting files.

Storage capacity is based on the number of tracks the drive is capable of recording. For example, 4–track drives can only record 20 MB of data on a 450 foot tape; 9–track drives can record up to 45 MB of data on a tape of the same length. QIC–11 is the only tape format available for 4–track tape drives. In contrast, 9–track tape drives can use either QIC–24 or QIC–11. Storage capacity is not appreciably affected by using

either format. QIC–24 is preferable to QIC–11 because it records a reference signal to mark the position of the first track on the tape, and each block has a unique block number.

The QIC–150 tape drives require DC–6150 (or equivalent) tape cartridges for writing. However, they can read other tape cartridges in QIC–11, QIC–24, QIC–120, or QIC–150 tape formats.

A number of additional **ioctl()** operations are available on "raw" devices. The following definitions are from **<sys/mtio.h>**:

```
/*
 * Structures and definitions for mag tape I/O control commands
 */

/*
 * Minor tape device definitions (except for ar).
 * Supports 2 units and 2-4 densities/formats per unit
 * depending on device.
 * Bits 0-1, unit number
 *      2, no rewind flag
 *    3-4, desity selection
 *    5-7, reserved
 */
#define MTUNIT(dev)     (minor(dev) & (4 -1))
#define MT_NOREWIND (1 <<2)
#define MT_DENSITY_MASK     (3 <<3)
#define MT_DENSITY1   (0 <<3)          /* Lowest density/format */
#define MT_DENSITY2   (1 <<3)
#define MT_DENSITY3   (2 <<3)
#define MT_DENSITY4   (3 <<3)          /* Highest density/format */


/* structure for MTIOCTOP – mag tape op command */
struct   mtop   {
        short    mt_op;          /* operations defined below */
        daddr_t mt_count;        /* how many of them */
};

#define  MTWEOF         0       /* write an end-of-file record */
#define  MTFSF          1       /* forward space file */
#define  MTBSF          2       /* backward space file */
#define  MTFSR          3       /* forward space record */
#define  MTBSR          4       /* backward space record */
#define  MTREW          5       /* rewind */
#define  MTOFFL         6       /* rewind and put the drive offline */
#define  MTNOP          7       /* no operation, sets status only */
#define  MTRETEN        8       /* retension the tape */
#define  MTERASE        9       /* erase the entire tape */
#define  MTEOM          10      /* position to end of media (SCSI only) */
#define  MTBSFM         11      /* backward space file mark */

/* structure for MTIOCGET – mag tape get status command */
struct   mtget {
        short    mt_type;                /* type of magtape device */
/* the following two registers are grossly device dependent */
        short    mt_dsreg;               /* "drive status" register */
```

```
        short   mt_erreg;              /* "error" register */
/* optional error info. */
        daddr_t mt_resid;              /* residual count */
        daddr_t mt_fileno;             /* file number of current position */
        daddr_t mt_blkno;              /* block number of current position */
};

/*
 * Constants for mt_type byte
 */
#define MT_ISTS                 0x01    /* vax: unibus ts-11 */
#define MT_ISHT                 0x02    /* vax: massbus tu77, etc */
#define MT_ISTM                 0x03    /* vax: unibus tm-11 */
#define MT_ISMT                 0x04    /* vax: massbus tu78 */
#define MT_ISUT                 0x05    /* vax: unibus gcr */
#define MT_ISCPC                0x06    /* sun: Multibus tapemaster */
#define MT_ISAR                 0x07    /* sun: Multibus archive */
#define MT_ISSC                 0x08    /* sun: SCSI archive */
#define MT_ISXY                 0x09    /* sun: Xylogics 472 */

#define MT_ISSYSGEN11           0x10    /* sun: SCSI Sysgen, QIC-11 only */
#define MT_ISSYSGEN             0x11    /* sun: SCSI Sysgen QIC-24/11 */
#define MT_ISDEFAULT            0x12    /* sun: SCSI default CCS */
#define MT_ISCCS3               0x13    /* sun: SCSI generic (unknown) CCS */
#define MT_ISMT02               0x14    /* sun: SCSI Emulex MT02 */
#define MT_ISVIPER1             0x15    /* sun: SCSI Archive QIC-150 Viper */
#define MT_ISWANGTEK1           0x16    /* sun: SCSI Wangtek QIC-150 */
#define MT_ISCCS7               0x17    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS8               0x18    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS9               0x19    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS11              0x1a    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS12              0x1b    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS13              0x1c    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS14              0x1d    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS15              0x1e    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS16              0x1f    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCDC                0x20    /* sun: SCSI CDC 1/2" cartridge */
#define MT_ISFUJI               0x21    /* sun: SCSI Fujitsu 1/2" cartridge */
#define MT_ISKENNEDY            0x22    /* sun: SCSI Kennedy 1/2" reel */
#define MT_ISHP                 0x23    /* sun: SCSI HP 1/2" reel */
#define MT_ISCCS21              0x24    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS22              0x25    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS23              0x26    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS24              0x27    /* sun: SCSI generic (unknown) CCS */
#define MT_ISEXABYTE            0x28    /* sun: SCSI Exabyte 8mm cartridge */
#define MT_ISCCS26              0x29    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS27              0x2a    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS28              0x2b    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS29              0x2c    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS30              0x2d    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS31              0x2e    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS32              0x2f    /* sun: SCSI generic (unknown) CCS */
```

```
/*
 * Device table structure and data for looking tape name from
 * tape id number.  Used by mt.c.
 */
struct mt_tape_info {
        short   t_type;         /* type of magtape device */
        char    *t_name;        /* printing name */
        char    *t_dsbits;      /* "drive status" register */
        char    *t_erbits;      /* "error" register */
};

#define MT_TAPE_INFO  { \
{ MT_ISCPC,             "TapeMaster 1/2-inch",          TMS_BITS, 0 }, \
{ MT_ISXY,              "Xylogics 472 1/2-inch", XTS_BITS, 0 }, \
{ MT_ISAR,              "Archive QIC-11",  ARCH_CTRL_BITS, ARCH_BITS }, \
{ MT_ISSYSGEN11,        "Sysgen QIC-11",                0, 0 }, \
{ MT_ISSYSGEN,          "Sysgen",                       0, 0 }, \
{ MT_ISMT02,            "Emulex MT-02 QIC-24",          0, 0 }, \
{ MT_ISVIPER1,          "Archive QIC-150",      0, 0 }, \
{ MT_ISWANGTEK1,        "Wangtek QIC-150",      0, 0 }, \
{ MT_ISKENNEDY,         "Kennedy 88780 1/2-inch",       0, 0 }, \
{ MT_ISHP,              "HP 88780 1/2-inch",    0, 0 }, \
{ MT_ISEXABYTE,         "Exabyte EXB-8200 8mm",         0, 0 }, \
{ 0 } \
}


/*
 * Constants for mt_type byte
 */

/*
 * Check if mt_type is one of the SCSI tape devices.
 */
#define MT_TYPE_SCSI(mt_type) \
        ((mt_type >= MT_ISSYSGEN11)  &&  (mt_type <= MT_ISCCS32))

/*
 * Older 1/4" cartridge tapes devices.
 * A blocking factor of 126 is recommended for compatibility.
 * A larger blocking factor may be used for improved streaming
 * performance.
 */
#define MT_TYPE_OLD_CARTRIDGE(mt_type) \
        ((mt_type == MT_ISSYSGEN11) || (mt_type == MT_ISSYSGEN) || \
         (mt_type == MT_ISAR))
```

```
/*
 * Current 1/4" cartridge tape devices.
 * A blocking factor of 40 (to 60) is recommended for
 * optimal streaming performance.
 */
#define MT_TYPE_NEW_CARTRIDGE(mt_type) \
        (mt_type >= MT_ISDEFAULT  &&  mt_type <= MT_ISCCS16)

/*
 * All 1/4" cartridge tape devices.
 * A blocking factor of 126 is recommended for compatibility
 * (during writes). See above for specific recommendations for
 * reading.
 */
#define MT_TYPE_CARTRIDGE(mt_type) \
        ((mt_type >= MT_ISSYSGEN11  &&  mt_type <= MT_ISCCS16)  || \
         (mt_type == MT_ISAR))

/*
 * All 1/2" reel tape devices.
 * A blocking factor of 20 is recommended for compatibility.
 */
#define MT_TYPE_REEL(mt_type) \
        ((mt_type >= MT_ISCDC  &&  mt_type <= MT_ISCCS32)  || \
         (mt_type >= MT_ISTS   &&  mt_type <= MT_ISCPC)    || \
         (mt_type == MT_ISXY))

/* mag tape I/O control commands */
#define  MTIOCTOP       _IOW(m, 1, struct mtop) /* do a mag tape op */
#define  MTIOCGET       _IOR(m, 2, struct mtget) /* get tape status */
#ifndef  KERNEL
#define  DEFTAPE        "/dev/rmt12"
#endif
```

SEE ALSO

mt(1), tar(1), read(2V), write(2V), ar(4S), st(4S), tm(4S), xt(4S)

BUGS

"Cooked" mode does not work for all magnetic tape devices (st, tm, and xt for example).

NAME

sd – driver for SCSI disk devices

CONFIG — SUN-3, SUN-3x and SUN-4 SYSTEMS

```
controller si0 at vme24d16 ? csr 0x200000 priority 2 vector siintr 0x40
controller si0 at obio ? csr 0x140000 priority 2
disk sd0 at si0 drive 0 flags 0
disk sd1 at si0 drive 1 flags 0
disk sd2 at si0 drive 8 flags 0
disk sd3 at si0 drive 9 flags 0
disk sd4 at si0 drive 16 flags 0
disk sd6 at si0 drive 24 flags 0

controller sc0 at vme24d16 ? csr 0x200000 priority 2 vector scintr 0x40
disk sd0 at sc0 drive 0 flags 0
disk sd1 at sc0 drive 1 flags 0
disk sd2 at sc0 drive 8 flags 0
disk sd3 at sc0 drive 9 flags 0
disk sd4 at sc0 drive 16 flags 0
disk sd6 at sc0 drive 24 flags 0
```

The first two **controller** lines above specify the first and second SCSI host adapters for Sun-3, Sun-3x, and Sun-4 VME systems. The third **controller** line specifies the first and only SCSI host adapter on Sun-3/50 and Sun-3/60 systems.

The four lines following the **controller** specification lines define the available **disk** devices, **sd0 – sd6i**.

The **flags** field is used to specify the SCSI device type to the host adapter. **flags** must be set to 0 to identify **disk** devices.

The **drive** value is the address emitted into the SCSI bus, to select a specific target and logical unit number. The value to enter in the **drive** field of the configuration line is calculated using the formula:

$$8 * target + lun$$

where *target* is the SCSI target, and *lun* is the SCSI logical unit number.

The next configuration block, following **si0** and **si1** above, describes the configuration for the older **sc0** host adapter. It uses the same configuration description as the **si0** host adapter.

CONFIG — SPARCsystem 330 and SUN-3/80 SYSTEMS

```
controller sm0 at obio ? csr 0xfa000000 priority 2
disk sd0 at sm0 drive 0 flags 0
disk sd1 at sm0 drive 1 flags 0
disk sd2 at sm0 drive 8 flags 0
disk sd3 at sm0 drive 9 flags 0
disk sd4 at sm0 drive 16 flags 0
disk sd6 at sm0 drive 24 flags 0
```

The SPARCsystem 330 and Sun-3/80 use an on-board SCSI host adapter, **sm0**. It follows the same rules as described above for the first CONFIG section.

CONFIG — SUN-4/110 SYSTEM

```
controller sw0 at obio 2 csr 0xa000000 priority 2
disk sd0 at sw0 drive 0 flags 0
disk sd1 at sw0 drive 1 flags 0
disk sd2 at sw0 drive 8 flags 0
disk sd3 at sw0 drive 9 flags 0
disk sd4 at sw0 drive 16 flags 0
disk sd6 at sw0 drive 24 flags 0
```

The Sun-4/110 uses an on-board SCSI host adapter, **sw0**. It follows the same rules as described above for the first CONFIG section.

CONFIG — SUN-2 SYSTEM

```
controller sc0 at mbmem ? csr 0x80000 priority 2
controller sc1 at mbmem ? csr 0x84000 priority 2
controller sc0 at vme24 ? csr 0x200000 priority 2 vector scintr 0x40
disk sd0 at sc0 drive 0 flags 0
disk sd1 at sc0 drive 1 flags 0
disk sd2 at sc0 drive 8 flags 0
disk sd2 at sc1 drive 0 flags 0
disk sd3 at sc1 drive 1 flags 0
```

The **controller** lines above specify the first and second SCSI host adapters, **sc0** and **sc1**, on Sun-2/120 and Sun-2/170 systems. The third **controller** line specifies the first and only host adapter on a Sun-2/160 system. It follows the same rules as described under the first CONFIG section above.

CONFIG — SUN-3/E SYSTEM

```
controller se0 at vme24d16 ? csr 0x300000 priority 2 vector se_intr 0x40
disk sd0 at se0 drive 0 flags 0
disk sd1 at se0 drive 1 flags 0
disk sd2 at se0 drive 8 flags 0
disk sd3 at se0 drive 9 flags 0
```

The Sun-3/E uses a VME-based SCSI host adapter, **se0**. It follows the same rules as described above for the first CONFIG section.

CONFIG — Sun386i

```
controller wds0 at obmem ? csr 0xFB000000 dmachan 7 irq 16 priority 2
disk sd0 at wds0 drive 0 flags 0
disk sd1 at wds0 drive 8 flags 0
disk sd2 at wds0 drive 16 flags 0
```

The Sun-386*i* configuration follows the same rules described above under the first CONFIG section.

DESCRIPTION

Files with minor device numbers **0-7** refer to various portions of drive 0. The standard device names begin with **sd** followed by the drive number and then a letter **a-h** for partitions **0-7** respectively. The character **?** stands here for a drive number in the range 0-6.

The block-files access the disk using the system's normal buffering mechanism and are read and written without regard to physical disk records. There is also a *raw* interface that provides for direct transmission between the disk and the user's read or write buffer. A single read or write call usually results in one I/O operation; raw I/O is therefore considerably more efficient when many bytes are transmitted. The names of the raw files conventionally begin with an extra **'r.'**

I/O requests (for example **lseek**(2)) to the SCSI disk must have an offset that is a multiple of 512 bytes (DEV_BSIZE), or the driver returns an EINVAL error. If the transfer length is not a multiple of 512 bytes, the transfer count is rounded up by the driver.

Disk Support

This driver handles the Adaptec ACB-4000 disk controller for ST-506 drives, the Emulex MD21 disk controller for ESDI drives, and embedded, CCS-compatible SCSI disk drives.

On Sun386*i* systems, this driver supports the CDC Wren III half-height, and Wren IV full-height SCSI disk drives.

The type of disk drive is determined using the SCSI inquiry command and reading the volume label stored on block 0 of the drive. The volume label describes the disk geometry and partitioning; it must be present or the disk cannot be mounted by the system.

The **sd?a** partition is normally used for the root file system on a disk, the **sd?b** partition as a paging area (for example, **swap**), and the **sd?c** partition for pack-pack copying. **sd?c** normally maps the entire disk and may also be used as the mount point for secondary disks in the system. The rest of the disk is normally the **sd?g** partition. For the primary disk, the user file system is located here.

FILES

/dev/sd[0-6][a-h]	block files
/dev/rsd[0-6][a-h]	raw files

SEE ALSO

dkio(4S), **directory**(3), **lseek**(2), **read**(2V), **write**(2V)

Product Specification for Wren IV SCSI Model 94171
Product Specification for Wren III SCSI Model 94161
Product Specification for Wren III SCSI Model 94211
Emulex MD21 Disk Controller Programmer Reference Manual
Adaptec ACB-4000 Disk Controller OEM Manual

DIAGNOSTICS

sd?: sdtimer: I/O request timeout
A tape I/O operation has taken too long to complete. A device or host adapter failure may have occurred.

sd?: sdtimer: can't abort request
The driver is unable to find the request in the disconnect que to notify the device driver that it has failed.

sd?: no space for inquiry data
sd?: no space for disk label
The driver was unable to get enough space for temporary storage. The driver is unable to open the disk device.

sd?: <%s>
The driver has found a SCSI disk device and opened it for the first time. The disk label is displayed to notify the user.

sd?: SCSI bus failure
A host adapter error was detected. The system may need to be rebooted.

sd?: single sector I/O failed
The driver attempted to recover from a transfer by writing each sector, one at a time, and failed. The disk needs to be reformatted to map out the new defect causing this error.

sd?: retry failed
sd?: rezero failed
A disk operation failed. The driver first tries to recover by retrying the command, if that fails, the driver rezeros the heads to cylinder 0 and repeats the retries. A failure of either the retry or rezero operations results in these warning messages; the error recovery operation continues until the retry count is exhausted. At that time a hard error is posted.

sd?: request sense failed
The driver was attempting to determine the cause of an I/O failure and was unable to get more information. This implies that the disk device may have failed.

sd?: warning, abs. block %d has failed %d times
The driver is warning the user that the specified block has failed repeatedly.

sd?: block %d needs mapping

sd?: reassigning defective abs. block %d

The specified block has failed repeatedly and may soon become an unrecoverable failure. If the driver does not map out the specified block automatically, it is recommend that the user correct the problem.

sd?: reassign block failed

The driver attempted to map out a block having excessive soft errors and failed. The user needs to run format and repair the disk.

sd?%c: cmd how blk %d (rel. blk %d)

sense key(0x%x): %s, error code(0x%x): %s An I/O operation (**cmd**), encountered an error condition at absolute block (**blk %d**), partition (**sd?%c:**), or relative block (**rel. block%d**). The error recovery operation (how) indicates whether it *retry*'ed, *restored*, or *failed*. The **sense key** and **error code** of the error are displayed for diagnostic purposes. The absolute **blk** of the the error is used for mapping out the defective block. The **rel. blk** is the block (sector) in error, relative to the beginning of the partition involved. This is useful for using **icheck**(8) to repair a damaged file structure on the disk.

NAME

sockio – ioctls that operate directly on sockets

SYNOPSIS

#include <sys/sockio.h>

DESCRIPTION

The IOCTL's listed in this manual page apply directly to sockets, independent of any underlying protocol. Note: the **setsockopt** system call (see **getsockopt**(2)) is the primary method for operating on sockets as such, rather than on the underlying protocol or network interface. **ioctl**s for a specific network interface or protocol are documented in the manual page for that interface or protocol.

SIOCSPGRP The argument is a pointer to an **int**. Set the process-group ID that will subsequently receive **SIGIO** or **SIGURG** signals for the socket referred to by the descriptor passed to **ioctl** to the value of that **int**.

SIOCGPGRP The argument is a pointer to an **int**. Set the value of that **int** to the process-group ID that is receiving **SIGIO** or **SIGURG** signals for the socket referred to by the descriptor passed to **ioctl**.

SIOCCATMARK The argument is a pointer to an **int**. Set the value of that **int** to 1 if the read pointer for the socket referred to by the descriptor passed to **ioctl** points to a mark in the data stream for an out-of-band message, and to 0 if it does not point to a mark.

SEE ALSO

ioctl(2), **getsockopt**(2), **filio**(4)

NAME

st – driver for SCSI tape devices

CONFIG — SUN-3, SUN-3x, SUN-4 SYSTEMS

controller si0 at vme24d16 ? csr 0x200000 priority 2 vector siintr 0x40
controller si1 at vme24d16 ? csr 0x204000 priority 2 vector siintr 0x41
controller si0 at obio ? csr 0x140000 priority 2
tape st0 at si0 drive 32 flags 1
tape st1 at si0 drive 40 flags 1
tape st2 at si1 drive 32 flags 1
tape st3 at si1 drive 40 flags 1

controller sc0 at vme24d16 ? csr 0x200000 priority 2 vector scintr 0x40
tape st0 at sc0 drive 32 flags 1
tape st1 at sc0 drive 40 flags 1

The first two **controller** lines above specify the first and second SCSI host adapters for Sun-3, Sun-3x, and Sun-4 VME systems. The third **controller** line specifies the first and only SCSI host adapter on Sun-3/50 and Sun-3/60 systems.

Following the **controller** specification lines are four lines which define the available **tape** devices, **st0–st3**. The first two **tape** devices, **st0** and **st1**, are on the first **controller**, **si0**. The next two **tape** devices, **st2** and **st3**, are on the second **controller**, **si1**.

The **flags** field is used to specify the SCSI device type to the host adapter. The **flags** field must be set to 1 to identify **tape** devices.

The **drive** value is calculated using the formula:

$8 * target + lun$

where *target* is the SCSI target, and *lun* is the SCSI logical unit number.

The next configuration block, following **si0** and **si1** above, describes the older **sc0** host adapter configuration. It follows the same configuration description as the **si0** host adapter.

CONFIG — SPARCsystem 330, SUN–3/80 SYSTEMS

controller sm0 at obio ? csr 0xfa000000 priority 2
tape st0 at sm0 drive 32 flags 1
tape st1 at sm0 drive 40 flags 1

The SPARCsystem 330 and Sun-3/80 use an on-board SCSI host adapter, **sm0**, which follows the rules described above in the Sun-3, Sun-3x, Sun-4 section.

CONFIG — SUN–4/110 SYSTEM

controller sw0 at obio 2 csr 0xa000000 priority 2
tape st0 at sw0 drive 32 flags 1
tape st1 at sw0 drive 40 flags 1

The Sun-4/110 uses an on-board SCSI host adapter, **sw0**, which follows the rules described above in the Sun-3, Sun-3x, Sun-4 section.

CONFIG — SUN–3/E SYSTEM

controller se0 at vme24d16 ? csr 0x300000 priority 2 vector se_intr 0x40
tape st0 at se0 drive 32 flags 1
tape st1 at se0 drive 40 flags 1

The Sun-3/E uses a VME-based SCSI host adapter, **se0**, which follows the rules described above for Sun-3, Sun-3x, Sun-4 systems.

CONFIG — Sun386i

controller wds0 at obmem ? csr 0xFB000000 dmachan 7 irq 16 priority 2
tape st0 at wds0 drive 32 flags 1

The Sun386*i* configuration follows the rules described above in the Sun-3, Sun-3x, Sun-4 configuration section.

CONFIG — SUN–2 SYSTEM

controller sc0 at mbmem ? csr 0x80000 priority 2
controller sc1 at mbmem ? csr 0x84000 priority 2
controller sc0 at vme24 ? csr 0x200000 priority 2 vector scintr 0x40
tape st0 at sc0 drive 32 flags 1
tape st1 at sc0 drive 40 flags 1
tape st0 at sc1 drive 32 flags 1
tape st1 at sc1 drive 40 flags 1

The **controller** lines above specify the first and second SCSI host adapters, **sc0** and **sc1**, on Sun-2/120 and Sun-2/170 systems. The third **controller** line specifies the only host adapter on a Sun-2/160. It follows the rules described in the Sun-3, Sun-3x, Sun-4 configuration section above.

DESCRIPTION

The **st** device driver is an interface to various SCSI tape devices. Supported 1/4–inch cartridge devices include the Archive Viper QIC–150 streaming tape drive, the Emulex MT–02 tape controller, and the Sysgen SC4000 tape controller. Supported 1/2–inch reel tape devices include the HP–88780 tape drive. **st** provides a standard interface to these various devices, see **mtio**(4) for details.

The driver can be opened with either rewind on close (**/dev/rst***) or no rewind on close (**/dev/nrst***) options. A maximum of four tape formats per device are supported (see FILES below). The tape format is specified using the device name. The four rewind on close formats for **st0**, for example, are **/dev/rst0**, **/dev/rst8**, **/dev/rst16**, and **/dev/rst24**.

Read Operation

Fixed-length I/O tape devices require the number of bytes read or written to be a multiple of the physical record size. For example, 1/4–inch cartridge tape devices only read or write multiples of 512 bytes.

Fixed-length tape devices read or write multiple records if the blocking factor is greater than 64512 bytes (minphys limit). These multiple writes are limited to 64512 bytes. For example, if a write request is issued for 65536 bytes using a 1/4–inch cartridge tape, two writes are issued; the first for 64512 bytes and the second for 1024 bytes.

Tape devices, which support variable-length I/O operations, such as 1/2–inch reel tape, may read or write a range of 1 to 65535 bytes. If the record size exceeds 65535 bytes, the driver reads or writes multiple records to satisfy the request. These multiple records are limited to 65534 bytes. As an example, if a write request for 65540 bytes is issued using 1/2–inch reel tape, two records are written; one for 65534 bytes followed by one for 6 bytes.

If the driver is opened for reading in a different format than the tape is written in, the driver overrides the user selected format. For example, if a 1/4–inch cartridge tape is written in QIC-24 format and opened for reading in QIC–11, the driver will detect a read failure on the first read and automatically switch to QIC–24 to recover the data.

Note: If the **/dev/*st[0–3]** format is used, no indication is given that the driver has overridden the user selected format. Other formats issue a warning message to inform the user of an overridden format selection. Some devices automatically perform this function and do not require driver support (1/2–inch reel and QIC–150 tape drives for example).

If a file mark is encountered during reading, no error is reported but the number of bytes transferred is zero. The next read operation reads into the next file.

End of media is indicated by two successive zero transfer counts. No further reading should be performed past the end of recorded media.

If the read request size is 2048 bytes, the tape driver behaves as a disk device and honors seek positioning requests (see **lseek**(2)). If a file mark is crossed during a read operation, this function is disabled.

Write Operation

Writing is allowed at either the beginning of tape or after the last written file on the tape. Writing from the beginning of tape is performed in the user-specified format. The original tape format is used for appending onto previously written tapes. A warning message is issued if the driver has to override the user-specified format.

Care should be used when appending files onto 1/2–inch reel tape devices, since an extra file mark is appended after the last file to mark the end of recorded media. In other words, the last file on the tape ends with two file marks instead of one. This extra file mark must be overwritten to prevent the creation of a null file. To facilitate write append operations, a space to the end of recorded media ioctl is provided to eliminate this problem by having the driver perform the positioning operation.

If the end of tape is encountered during writing, no error is reported but the number of bytes transferred is zero and no further writing is allowed. Trailer records may be written by first writing a file mark followed by the trailer records. It is important that these trailer records be kept as short as possible to prevent data loss.

Close Operation

If data was written, a file mark is automatically written by the driver upon close. If the rewinding device name is used, the tape will be rewound after the file mark is written. If the user wrote a file mark prior to closing, then no file mark is written upon close. If a file positioning ioctl, like rewind, is issued after writing, a file mark is written before repositioning the tape.

Note: For 1/2–inch reel tape devices, two file marks are written to mark the end of recorded media before rewinding or performing a file positioning ioctl. If the user wrote a file mark before closing a 1/2–inch reel tape device, the driver will always write a file mark before closing to insure that the end of recorded media is marked properly.

If no data was written and the driver was opened for WRITE-ONLY access, a file mark is written thus creating a null file.

IOCTLS

The following ioctls are supported: forwardspace record, forwardspace file, backspace record, backspace file, backspace file mark, rewind, write file mark, offline, erase, retension, space to EOM, and get status.

The backspace file and forwardspace file tape operations are inverses. Thus, a forwardspace "–1" file is equivalent to a backspace "1" file. A backspace "0" file is the same as forwardspace "0" file; both position the tape device to the beginning of the current file.

Backspace file mark moves the tape backwards by file marks. The tape position will end on the beginning of tape side of the desired file mark. Devices which do not support this function, such as 1/4–inch cartridge tape, return an ENXIO error.

Backspace record and forwardspace record operations perform much like space file operations, except that they move by records instead of files. Variable-length I/O devices (1/2–inch reel, for example) space actual records; fixed-length I/O devices space physical records (blocks). 1/4–inch cartridge tape, for example, spaces 512 byte physical records. The status ioctl residue count contains the number of files or records not skipped. Record skipping does not go past a file mark; file skipping does not go past the end of recorded media.

Spacing to the end of recorded media positions the tape at a location just after the last file written on the tape. For 1/4–inch cartridge tape, this is after the last file mark on the tape. For 1/2–inch reel tape, this is just after the first file mark but before the second (and last) file mark on the tape. Additional files can then be appended onto the tape from that point.

The offline ioctl rewinds and, if appropriate, takes the device offline by unloading the tape. Tape must be inserted before the tape device can be used again.

The erase ioctl rewinds the tape, erases it completely, and returns to the beginning of tape.

The retension ioctl only applies to 1/4–inch cartridge tape devices. It is used to restore tape tension improving the tape's soft error rate after extensive start-stop operations or long-term storage. Devices which do not support this function, such as 1/2–inch reel tape, return an ENXIO error.

The get status ioctl call returns the drive id (mt_type), sense key error (mt_erreg), file number (mt_fileno), and record number (mt_blkno) of the last error. The residue count (mt_resid) is set to the number of bytes not transferred or files/records not spaced.

Note: The error status is reset by the get status ioctl call or the next read, write, or other ioctl operation. If no error has occurred (sense key is zero), the current file and record position are returned.

ERRORS

EACCES	The driver is opened for write access and the tape is write protected, or an attempt is made to write on a write protected tape. For writing with QIC–150 tape drives, this error is also reported if the wrong tape media is used for writing.
EBUSY	The tape device is already in use.
EIO	During opening, the tape device is not ready because either no tape is in the drive, or the drive is not on-line. Once open, this error is returned if the requested I/O transfer could not be completed.
EINVAL	The number of bytes read or written is not a multiple of the physical record size (fixed-length tape devices only).
ENXIO	During opening, the tape device does not exist. On ioctl functions, this indicates that the tape device does not support the ioctl function.

FILES

For 1/2–inch reel tape devices (HP-88780):

/dev/rst[0–3]	800 BPI density
/dev/rst[8–11]	1600 BPI density
/dev/rst[16–20]	6250 BPI density
/dev/rst[24–28]	data compression
/dev/nrst[0–3]	non–rewinding 800 BPI density
/dev/nrst[8–11]	non–rewinding 1600 BPI density
/dev/nrst[16–19]	non–rewinding 6250 density
/dev/nrst[24–27]	non–rewinding data compression

For helical-scan tape devices (Exabyte):

/dev/rst[0–3]	Standard Format
/dev/rst[8–11]	Standard Format
/dev/rst[16–20]	Standard Format
/dev/rst[24–28]	Standard Format
/dev/nrst[0–3]	non-rewinding Standard Format
/dev/nrst[8–11]	non-rewinding Standard Format
/dev/nrst[16–19]	non-rewinding Standard Format
/dev/nrst[24–27]	non-rewinding Standard Format

For QIC–150 tape devices (Archive Viper):

/dev/rst[0–3]	QIC–150 Format
/dev/rst[8–11]	QIC–150 Format
/dev/rst[16–20]	QIC–150 Format
/dev/rst[24–28]	QIC–150 Format
/dev/nrst[0–3]	non-rewinding QIC–150 Format
/dev/nrst[8–11]	non-rewinding QIC–150 Format

/dev/nrst[16–19] non-rewinding QIC–150 Format
/dev/nrst[24–27] non-rewinding QIC–150 Format

For QIC–24 tape devices (Emulex MT–02 and Sysgen SC4000):

/dev/rst[0–3] QIC–11 Format
/dev/rst[8–11] QIC–24 Format
/dev/rst[16–20] QIC–24 Format
/dev/rst[24–28] QIC–24 Format
/dev/nrst[0–3] non-rewinding QIC–11 Format
/dev/nrst[8–11] non-rewinding QIC–24 Format
/dev/nrst[16–19] non-rewinding QIC–24 Format
/dev/nrst[24–27] non-rewinding QIC–24 Format

Note: The QIC–24 format is preferred over QIC–11 for Sun-3, Sun-3x, Sun-4, and Sun386*i* systems.

SEE ALSO

mt(1), **tar**(1), **mtio**(4), **dump**(8), **restore**(8)

Hewlett–Packard HP–88780 Tape Drive Product Specification
Archive Viper QIC–150 Tape Drive Product Specification
Emulex MT–02 Intelligent Tape Controller Product Specification
Sysgen SC4000 Intelligent Tape Controller Product Specification

DIAGNOSTICS

st?: sttimer: I/O request timeout
A tape I/O operation has taken too long to complete. A device or host adapter failure may have occurred.

st?: sttimer: can't abort request
The driver is unable to find the request in the disconnect que to notify the device driver that it has failed. A SCSI bus reset is issued to recover from this error.

st?: unknown SCSI device found
The SCSI device is not a tape device; it is some other type of SCSI device.

st?: warning, unknown tape drive found
The driver does not recognize the tape device. Only the default tape density is used; block size is set to the value specified by the tape drive.

st?: tape is write protected
The tape is write protected.

st?: wrong tape for writing, use DC6150 tape (or equivalent)
For QIC–150 tape drives, this indicates that the user is trying to write on a DC–300XL (or equivalent) tape. Only DC–6150 (or equivalent) tapes can be used for writing.
Note: DC–6150 was formerly known as DC–600XTD.

st?: warning, rewinding tape
The driver is rewinding tape in order to set the tape format.

st?: warning, using alternate tape format
The driver is overriding the user-selected tape format and using the previously used format.

st?: warning, tape rewound
For Sysgen tape controllers, the tape may be rewound as a result of getting sense data.

st?: format change failed
The tape drive rejected the mode select command to change the tape format.

st?: file mark write failed
The driver was unable to write a file mark.

st?: warning, The tape may be wearing out or the head may need cleaning.
st?: read retries= %d, file= %d, block= %d
st?: write retries= %d, file= %d, block= %d
The number of allowable soft errors has been exceeded for this tape. Either the tape heads need cleaning or the tape is wearing out. If the tape is wearing out, continued usage of it is not recommended.

st?: illegal command
The SCSI command just issued was illegal. This message can result from issuing an inappropriate command, such as trying to write over previously written files on the tape. On foreign tape devices, this can also be caused by selecting the wrong tape format.

st?: error: sense key(0x%x): %s, error code(0x%x): %s
An error has occurred. The sense key message and error code are displayed for diagnostic purposes.

st?: stread: not modulo %d block size
st?: stwrite: not modulo %d block size
The read or write request size must be a multiple of the **%d** physical block size.

st?: file positioning error
st?: block positioning error
The driver was unable to position the tape to the desired file or block (record). This is probably caused by a damaged tape.

BUGS

Foreign tape devices which do not return a BUSY status during tape loading prevent user commands from being held until the device is ready. The user must delay issuing any tape operations until the tape device is ready. This is not a problem for Sun supplied tape devices.

Foreign tape devices which do not report a blank check error at the end of recorded media cause file positioning operations to fail. Some tape drives for example, mistakenly report media error instead of blank check error.

"Cooked" mode for read and write operations is not supported.

Systems using the older **sc0** host adapter or the Sysgen SC4000 tape controller, prevent disk I/O over the SCSI bus while the tape is in use (during a rewind for example). This problem is caused by the fact that they do not support disconnect/reconnect to free the SCSI bus. Newer tape devices, like the the Emulex MT–02, and host adapters, like **si0**, eliminate this problem.

Some older systems may not support the QIC–24 format, and may complain (or exhibit erratic behavior) when the user attempts to use this format.

NAME

streamio – STREAMS ioctl commands

SYNOPSIS

#include <stropts.h>
int ioctl (fildes, command, arg)
int fildes, command;

DESCRIPTION

STREAMS (see **intro**(2)) ioctl commands are a subset of **ioctl**(2) commands that perform a variety of control functions on STREAMS. The arguments *command* and *arg* are passed to the file designated by *fildes* and are interpreted by the *stream*head. Certain combinations of these arguments may be passed to a module or driver in the stream.

fildes is an open file descriptor that refers to a stream. *command* determines the control function to be performed as described below. *arg* represents additional information that is needed by this command. The type of *arg* depends upon the command, but it is generally an integer or a pointer to a *command*-specific data structure.

Since these STREAMS commands are a subset of *ioctl*, they are subject to the errors described there. In addition to those errors, the call will fail with *errno* set to EINVAL, without processing a control function, if the stream referenced by *fildes* is linked below a multiplexor, or if *command* is not a valid value for a *stream*.

Also, as described in *ioctl*, STREAMS modules and drivers can detect errors. In this case, the module or driver sends an error message to the *stream head* containing an error value. Subsequent system calls will fail with *errno* set to this value.

IOCTLS

The following **ioctl** commands, with error values indicated, are applicable to all STREAMS files:

I_PUSH Pushes the module whose name is pointed to by *arg* onto the top of the current stream, just below the *stream*head. It then calls the open routine of the newly-pushed module.

I_PUSH will fail if one of the following occurs:

EINVAL	The module name is invalid.
EFAULT	*arg* points outside the allocated address space.
ENXIO	The open routine of the new module failed.
ENXIO	A hangup is received on the stream referred to by *fildes*.

I_POP Removes the module just below the *stream head* of the stream pointed to by *fildes*. *arg* should be 0 in an **I_POP** request.

I_POP will fail if one of the following occurs:

EINVAL	No module is present on *stream*.
ENXIO	A hangup is received on the stream referred to by *fildes*.

I_LOOK Retrieves the name of the module just below the *stream head* of the stream pointed to by *fildes*, and places it in a NULL terminated character string pointed at by *arg*. The buffer pointed to by *arg* should be at least **FMNAMESZ+1** bytes long. An '**#include <sys/conf.h>**' declaration is required.

I_LOOK will fail if one of the following occurs:

EFAULT	*arg* points outside the allocated address space of the process.
EINVAL	No module is present on *stream*.

I_FLUSH This request flushes all input and/or output queues, depending on the value of *arg*. Legal *arg* values are:

FLUSHR	Flush read queues.
FLUSHW	Flush write queues.
FLUSHRW	Flush read and write queues.

I_FLUSH will fail if one of the following occurs:

EAGAIN	No buffers could be allocated for the flush message.
EINVAL	The value of *arg* is invalid.
ENXIO	A hangup is received on the stream referred to by *fildes*.

I_SETSIG Informs the *stream head* that the user wishes the kernel to issue the **SIGPOLL** signal (see **sigvec**(2)) when a particular event has occurred on the stream associated with *fildes*. **I_SETSIG** supports an asynchronous processing capability in STREAMS. The value of *arg* is a bitmask that specifies the events for which the user should be signaled. It is the bitwise-OR of any combination of the following constants:

S_INPUT	A non-priority message has arrived on a *stream head* read queue, and no other messages existed on that queue before this message was placed there. This is set even if the message is of zero length.
S_HIPRI	A priority message is present on the *stream head* read queue. This is set even if the message is of zero length.
S_OUTPUT	The write queue just below the *stream head* is no longer full. This notifies the user that there is room on the queue for sending (or writing) data downstream.
S_MSG	A STREAMS signal message that contains the **SIGPOLL** signal has reached the front of the *stream head* read queue.

A user process may choose to be signaled only of priority messages by setting the *arg* bitmask to the value **S_HIPRI**.

Processes that wish to receive **SIGPOLL** signals must explicitly register to receive them using **I_SETSIG**. If several processes register to receive this signal for the same event on the same *stream*, each process will be signaled when the event occurs.

If the value of *arg* is zero, the calling process will be unregistered and will not receive further **SIGPOLL** signals.

I_SETSIG will fail if one of the following occurs:

EINVAL	The value of *arg* is invalid or *arg* is zero and the process is not registered to receive the **SIGPOLL** signal.
EAGAIN	A data structure could not be allocated to store the signal request.

I_GETSIG Returns the events for which the calling process is currently registered to be sent a **SIGPOLL** signal. The events are returned as a bitmask pointed to by *arg*, where the events are those specified in the description of **I_SETSIG** above.

I_GETSIG will fail if one of the following occurs:

EINVAL	The process is not registered to receive the **SIGPOLL** signal.
EFAULT	*arg* points outside the allocated address space of the process.

I_FIND This request compares the names of all modules currently present in the stream to the name pointed to by *arg*, and returns 1 if the named module is present in the stream. It returns 0 if the named module is not present.

I_FIND will fail if one of the following occurs:

EFAULT	*arg* points outside the allocated address space of the process.
EINVAL	*arg* does not point to a valid module name.

I_PEEK This request allows a user to retrieve the information in the first message on the *stream head* read queue without taking the message off the queue. *arg* points to a *strpeek* structure which contains the following members:

```
struct strbuf   ctlbuf;
struct strbuf   databuf;
long            flags;
```

The *maxlen* field in the *ctlbuf* and *databuf strbuf* structures (see **getmsg**(2)) must be set to the number of bytes of control information and/or data information, respectively, to retrieve. If the user sets *flags* to **RS_HIPRI, I_PEEK** will only look for a priority message on the *stream head* read queue.

I_PEEK returns 1 if a message was retrieved, and returns 0 if no message was found on the *stream head* read queue, or if the **RS_HIPRI** flag was set in *flags* and a priority message was not present on the *stream head* read queue. It does not wait for a message to arrive. On return, *ctlbuf* specifies information in the control buffer, *databuf* specifies information in the data buffer, and *flags* contains the value 0 or **RS_HIPRI.**

I_PEEK will fail if one of the following occurs:

EFAULT	*arg* points, or the buffer area specified in *ctlbuf* or *databuf* is, outside the allocated address space of the process.

I_SRDOPT Sets the read mode using the value of the argument *arg*. Legal *arg* values are:

RNORM	Byte-stream mode, the default.
RMSGD	Message-discard mode.
RMSGN	Message-nondiscard mode.

Read modes are described in **read**(2V).

I_SRDOPT will fail if one of the following occurs:

EINVAL	*arg* is not one of the above legal values.

I_GRDOPT Returns the current read mode setting in an *int* pointed to by the argument *arg*. Read modes are described in **read**(2V).

I_GRDOPT will fail if one of the following occurs:

EFAULT	*arg* points outside the allocated address space of the process.

I_NREAD Counts the number of data bytes in data blocks in the first message on the *stream head* read queue, and places this value in the location pointed to by *arg*. The return value for the command is the number of messages on the *stream head* read queue. For example, if zero is returned in *arg*, but the **ioctl** return value is greater than zero, this indicates that a zero-length message is next on the queue.

I_NREAD will fail if one of the following occurs:

EFAULT *arg* points outside the allocated address space of the process.

I_FDINSERT creates a message from user specified buffer(s), adds information about another stream and sends the message downstream. The message contains a control part and an optional data part. The data and control parts to be sent are distinguished by placement in separate buffers, as described below.

arg points to a *strfdinsert* structure which contains the following members:

```
struct strbuf   ctlbuf;
struct strbuf   databuf;
long            flags;
int             fd;
int             offset;
```

The *len* field in the *ctlbuf strbuf* structure (see **putmsg**(2)) must be set to the size of a pointer plus the number of bytes of control information to be sent with the message. *fd* specifies the file descriptor of the other stream and *offset*, which must be word-aligned, specifies the number of bytes beyond the beginning of the control buffer where **I_FDINSERT** will store a pointer to the *fd* stream's driver read queue structure. The *len* field in the *databuf strbuf* structure must be set to the number of bytes of data information to be sent with the message or zero if no data part is to be sent.

flags specifies the type of message to be created. A non-priority message is created if *flags* is set to 0, and a priority message is created if *flags* is set to **RS_HIPRI**. For non-priority messages, **I_FDINSERT** will block if the stream write queue is full due to internal flow control conditions. For priority messages, **I_FDINSERT** does not block on this condition. For non-priority messages, **I_FDINSERT** does not block when the write queue is full and **O_NDELAY** is set. Instead, it fails and sets *errno* to EAGAIN.

I_FDINSERT also blocks, unless prevented by lack of internal resources, waiting for the availability of message blocks in the *stream*, regardless of priority or whether **O_NDELAY** has been specified. No partial message is sent.

I_FDINSERT will fail if one of the following occurs:

EAGAIN A non-priority message was specified, the **O_NDELAY** flag is set, and the stream write queue is full due to internal flow control conditions.

EAGAIN Buffers could not be allocated for the message that was to be created.

EFAULT *arg* points, or the buffer area specified in *ctlbuf* or *databuf* is, outside the allocated address space of the process.

EINVAL *fd* in the *strfdinsert* structure is not a valid, open stream file descriptor; the size of a pointer plus *offset* is greater than the *len* field for the buffer specified through *ctlptr*;

offset does not specify a properly-aligned location in the data buffer; an undefined value is pointed to by *flags*.

ENXIO — A hangup is received on the stream referred to by *fildes*.

ERANGE — The *len* field for the buffer specified through *databuf* does not fall within the range specified by the maximum and minimum packet sizes of the topmost stream module, or the *len* field for the buffer specified through *databuf* is larger than the maximum configured size of the data part of a message, or the *len* field for the buffer specified through *ctlbuf* is larger than the maximum configured size of the control part of a message.

I_STR Constructs an internal STREAMS ioctl message from the data pointed to by *arg*, and sends that message downstream.

This mechanism is provided to permit a process to specify timeouts and variable-sized amounts of data when sending an **ioctl** request to downstream modules and drivers. It allows information to be sent with the *ioctl*, and will return to the user any information sent upstream by the downstream recipient. **I_STR** blocks until the system responds with either a positive or negative acknowledgement message, or until the request "times out" after some period of time. If the request times out, it fails with *errno* set to ETIME.

At most, one **I_STR** can be active on a stream. Further **I_STR** calls will block until the active **I_STR** completes at the *stream head*. The default timeout interval for these requests is 15 seconds. The **O_NDELAY** (see **open**(2V)) flag has no effect on this call.

To send requests downstream, *arg* must point to a *strioctl* structure which contains the following members:

```
int     ic_cmd;     /* downstream command */
int     ic_timout;  /* ACK/NAK timeout */
int     ic_len;     /* length of data arg */
char    *ic_dp;     /* ptr to data arg */
```

ic_cmd is the internal ioctl command intended for a downstream module or driver and *ic_timout* is the number of seconds (−1 = infinite, 0 = use default, >0 = as specified) an **I_STR** request will wait for acknowledgement before timing out. *ic_len* is the number of bytes in the data argument and *ic_dp* is a pointer to the data argument. The *ic_len* field has two uses: on input, it contains the length of the data argument passed in, and on return from the command, it contains the number of bytes being returned to the user (the buffer pointed to by *ic_dp* should be large enough to contain the maximum amount of data that any module or the driver in the stream can return).

The *stream head* will convert the information pointed to by the *strioctl* structure to an internal ioctl command message and send it downstream.

I_STR will fail if one of the following occurs:

EAGAIN — Buffers could not be allocated for the **ioctl** message.

EFAULT — *arg* points, or the buffer area specified by *ic_dp* and *ic_len* (separately for data sent and data returned) is, outside the allocated address space of the process.

EINVAL — *ic_len* is less than 0 or *ic_len* is larger than the maximum

configured size of the data part of a message or *ic_timout* is less than −1.

ENXIO — A hangup is received on the stream referred to by *fildes*.

ETIME — A downstream **ioctl** timed out before acknowledgement was received.

An **I_STR** can also fail while waiting for an acknowledgement if a message indicating an error or a hangup is received at the *stream*head. In addition, an error code can be returned in the positive or negative acknowledgement message, in the event the **ioctl** command sent downstream fails. For these cases, **I_STR** will fail with *errno* set to the value in the message.

I_SENDFD Requests the stream associated with *fildes* to send a message, containing a file pointer, to the *stream head* at the other end of a stream pipe. The file pointer corresponds to *arg*, which must be an integer file descriptor.

I_SENDFD converts *arg* into the corresponding system file pointer. It allocates a message block and inserts the file pointer in the block. The user id and group id associated with the sending process are also inserted. This message is placed directly on the read queue (see **intro**(2)) of the *stream head* at the other end of the stream pipe to which it is connected.

I_SENDFD will fail if one of the following occurs:

EAGAIN — The sending stream is unable to allocate a message block to contain the file pointer.

EAGAIN — The read queue of the receiving *stream head* is full and cannot accept the message sent by **I_SENDFD**.

EBADF — *arg* is not a valid, open file descriptor.

EINVAL — *fildes* is not connected to a stream pipe.

ENXIO — A hangup is received on the stream referred to by *fildes*.

I_RECVFD Retrieves the file descriptor associated with the message sent by an **I_SENDFD** **ioctl** over a stream pipe. *arg* is a pointer to a data buffer large enough to hold an *strrecvfd* data structure containing the following members:

```
int fd;
unsigned short uid;
unsigned short gid;
char fill[8];
```

fd is an integer file descriptor. *uid* and *gid* are the user ID and group ID, respectively, of the sending stream.

If **O_NDELAY** is not set (see **open**(2V)), **I_RECVFD** will block until a message is present at the *stream*head. If **O_NDELAY** is set, **I_RECVFD** will fail with *errno* set to EAGAIN if no message is present at the *stream*head.

If the message at the *stream head* is a message sent by an **I_SENDFD**, a new user file descriptor is allocated for the file pointer contained in the message. The new file descriptor is placed in the *fd* field of the *strrecvfd* structure. The structure is copied into the user data buffer pointed to by *arg*.

I_RECVFD will fail if one of the following occurs:

EAGAIN — A message was not present at the *stream head* read queue, and the **O_NDELAY** flag is set.

EBADMSG	The message at the *stream head* read queue was not a message containing a passed file descriptor.
EFAULT	*arg* points outside the allocated address space of the process.
EMFILE	Too many descriptors are active.
ENXIO	A hangup is received on the stream referred to by *fildes*.

The following four commands are used for connecting and disconnecting multiplexed STREAMS configurations.

I_LINK Connects two streams, where *fildes* is the file descriptor of the stream connected to the multiplexing driver, and *arg* is the file descriptor of the stream connected to another driver. The stream designated by *arg* gets connected below the multiplexing driver. **I_LINK** causes the multiplexing driver to send an acknowledgement message to the *stream head* regarding the linking operation. This call returns a multiplexor ID number (an identifier used to disconnect the multiplexor, see **I_UNLINK**) on success, and a −1 on failure.

I_LINK will fail if one of the following occurs:

ENXIO	A hangup is received on the stream referred to by *fildes*.
ETIME	The **ioctl** timed out before an acknowledgement was received.
EAGAIN	Storage could not be allocated to perform the **I_LINK**.
EBADF	*arg* is not a valid, open file descriptor.
EINVAL	The stream referred to by *fildes* does not support multiplexing.
EINVAL	*arg* is not a stream, or is already linked under a multiplexor.
EINVAL	The specified link operation would cause a "cycle" in the resulting configuration; that is, if a given *stream head* is linked into a multiplexing configuration in more than one place.

An **I_LINK** can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the *stream head* of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, **I_LINK** will fail with *errno* set to the value in the message.

I_UNLINK Disconnects the two streams specified by *fildes* and *arg*. *fildes* is the file descriptor of the stream connected to the multiplexing driver. *arg* is the multiplexor ID number that was returned by the **ioctl I_LINK** command when a stream was linked below the multiplexing driver. If *arg* is −1, then all streams which were linked to *fildes* are disconnected. As in **I_LINK**, this command requires the multiplexing driver to acknowledge the unlink.

I_UNLINK will fail if one of the following occurs:

ENXIO	A hangup is received on the stream referred to by *fildes*.
ETIME	The **ioctl** timed out before an acknowledgement was received.
EAGAIN	Buffers could not be allocated for the acknowledgement message.

EINVAL The multiplexor ID number was invalid.

An **I_UNLINK** can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the *stream head* of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, **I_UNLINK** will fail with *errno* set to the value in the message.

SEE ALSO

close(2), **fcntl**(2V), **getmsg**(2), **intro**(2), **ioctl**(2), **open**(2V), **poll**(2), **putmsg**(2), **read**(2V), **sigvec**(2), **write**(2V)

STREAMS Programmer's Guide
STREAMS Primer

NAME

taac – Sun applications accelerator

CONFIG

taac0 at vme32d32 ? csr 0x28000000

DESCRIPTION

The **taac** interface supports the optional TAAC-1 Applications Accelerator. This add-on device is composed of a very-long-instruction-word computation engine, coupled with an 8MB memory array. This memory area can be used either as a frame buffer, or as storage for large data sets.

Programs can be downloaded for execution on the TAAC-1 directly, they can be executed by the host processor, or the host processor and the TAAC-1 engine can be used in combination. See the *TAAC-1 User's Guide* for detailed information on accessing the TAAC-1 from the host. This manual also describes the C compiler, the programming tools, and the support libraries for the TAAC-1.

Programs on the host processor gain access to the TAAC-1 registers and memory by using **mmap**(2).

FILES

/dev/taac0
/usr/include/taac1
/usr/lib/taac1

SEE ALSO

mmap(2)

TAAC-1 Application Accelerator: User Guide

NAME

comsat, in.comsat – biff server

SYNOPSIS

/usr/etc/in.comsat

DESCRIPTION

comsat is the server process which listens for reports of incoming mail and notifies users who have requested to be told when mail arrives. It is invoked as needed by **inetd**(8C), and times out if inactive for a few minutes.

comsat listens on a datagram port associated with the **biff**(1) service specification (see **services**(5)) for one line messages of the form

user@mailbox-offset

If the *user* specified is logged in to the system and the associated terminal has the owner execute bit turned on (by a '**biff y**'), the *offset* is used as a seek offset into the appropriate mailbox file and the first 7 lines or 560 characters of the message are printed on the user's terminal. Lines which appear to be part of the message header other than the **From**, **To**, **Date**, or **Subject** lines are not printed when displaying the message.

FILES

/etc/utmp	to find out who's logged on and on what terminals

SEE ALSO

biff(1), **services**(5), **inetd**(8C)

BUGS

The message header filtering is prone to error.

The notification should appear in a separate window so it does not mess up the screen.

NAME

config – build system configuration files

SYNOPSIS

/usr/etc/config [**–fgnp**] [**–o** *obj_dir*] *config_file*

DESCRIPTION

config does the preparation necessary for building a new system kernel with **make**(1). The *config_file* named on the command line describes the kernel to be made in terms of options you want in your system, size of tables, and device drivers to be included. When you run **config**, it uses several input files located in the current directory (typically the **conf** subdirectory of the system source including your *config_file*). The format of this file is described below.

If the directory named *../config_file* does not exist, **config** will create one. One of **config**'s output files is a makefile which you use with **make**(1) to build your system.

You use **config** as follows. Run **config** from the **conf** subdirectory of the system source (in a typical Sun environment, from **/usr/share/sys/sun[2 3 4]/conf**):

```
example# /usr/etc/config config_file
Doing a "make depend"
example# cd ../config_file
example# make
...lots of output...
```

While **config** is running watch for any errors. Never use a kernel which **config** has complained about; the results are unpredictable. If **config** completes successfully, you can change directory to the *../config_file* directory, where it has placed the new makefile, and use **make** to build a kernel. The output files placed in this directory include **ioconf.c**, which contains a description of I/O devices attached to the system; **mbglue.s**, which contains short assembly language routines used for vectored interrupts, a makefile, which is used by **make** to build the system; a set of header files (*device_name*.**h**) which contain the number of various devices that may be compiled into the system; and a set of swap configuration files which contain definitions for the disk areas to be used for the root file system, swapping, and system dumps.

Now you can install your new kernel and try it out.

OPTIONS

–f Set up the makefile for fast builds. This is done by building a **vmunix.o** file which includes all the **.o** files which have no source. This reduces the number of files which have to be **stated** during a system build. This is done by prelinking all the files for which no source exists into another file which is then linked in place of all these files when the kernel is made. This makefile is faster because it does not **stat** the object files during the build.

–g Get the current version of a missing source file from its SCCS history, if possible.

–n Do not do the **'make depend'**. Normally **config** will do the **'make depend'** automatically. If this option is used **config** will print **'Don't forget to do a "make depend"'** before completing as a reminder.

–p Configure the system for profiling (see **kgmon**(8) and **gprof**(1)).

–o *obj_dir*
Use *../obj_dir* instead of **../OBJ** as the directory to find the object files when the corresponding source file is not present in order to generate the files necessary to compile and link your kernel.

USAGE

Input Grammar

In the following descriptions, a number can be a decimal integer, a whole octal number or a whole hexadecimal number. Hex and octal numbers are specified to **config** in the same way they are specified to the C compiler, a number starting with **0x** is a hex number and a number starting with just a **0** is an octal number.

Comments are begin with a # character, and end at the next NEWLINE. Lines beginning with TAB characters are considered continuations of the previous line. Lines of the configuration file can be one of two basic types. First, there are lines which describe general things about your system:

machine "*type***"**

This is system is to run on the machine type specified. Only one machine type can appear in the config file. The legal *type*s for a Sun system are **sun2, sun3, sun4,** and **sun386**. Note: the double quotes around *type* are part of the syntax, and must be included.

cpu "*type***"**

This system is to run on the cpu type specified. More than one cpu type can appear in the config file. Legal *type*s for a **sun2** machine are noted in the annotated config file in *Installing SunOS 4.1*.

ident *name*

Give the system identifier — a name for the machine or machines that run this kernel. Note that *name* must be enclosed in double quotes if it contains both letters and digits. Also, note that if *name* is **GENERIC**, you need not include the '**options GENERIC**' clause in order to specify '**swap generic**'.

maxusers *number*

The maximum expected number of simultaneously active user on this system is *number*. This number is used to size several system data structures.

options *optlist*

Compile the listed options into the system. Options in this list are separated by commas. A line of the form:

options FUNNY,HAHA

yields

–DFUNNY –DHAHA

to the C compiler. An option may be given a value, by following its name with = (equal sign) then the value enclosed in (double) quotes. None of the standard options use such a value.

In addition, options can be used to bring in additional files if the option is listed in the **files** files. All options should be listed in upper case. In this case, no corresponding *option*.**h** will be created as it would be using the corresponding *pseudo-device* method.

config *sysname config_clauses*...

Generate a system with name *sysname* and configuration as specified in *config-clauses*. The *sysname* is used to name the resultant binary image and per-system swap configuration files. The *config_clauses* indicate the location for the root file system, one or more disk partitions for swapping and paging, and a disk partition to which system dumps should be made. All but the root device specification may be omitted; **config** will assign default values as described below.

root A root device specification is of the form '**root on** *xy0d*'. If a specific partition is omitted — for example, if only **root on xy0** is specified — the '**a**' partition is assumed. When a generic system is being built, no root specification should be given; the root device will be defined at boot time by prompting the console.

swap To specify a swap partition, use a clause of the form: '**swap on** *partition*'. Swapping areas may be almost any size. Partitions used for swapping are sized at boot time by the system; to override dynamic sizing of a swap area the number of sectors in the swap area can be specified in the config file. For example, '**swap on** *xy0b* **size 99999**' would configure a swap partition with 99999 sectors. If **swap generic** or no *partition* is specified with **on**, partition b on the root device is used. For dataless clients, use '**swap on type nfs**'.

To configure multiple swap partitions, specify multiple '**swap on**' clauses. For example:

config vmunix swap on xy0 swap on xy1

dumps The location to which system dumps are sent may be specified with a clause of the form '**dumps on** *xy1*'. If no dump device is specified, the first swap partition specified is used. If a device is specified without a particular partition, the '**b**' partition is assumed. If a generic configuration is to be built, no dump device should be specified; the dump device will be assigned to the swap device dynamically configured at boot time. Dumps are placed at the end of the partition specified. Their size and location is recorded in global kernel variables *dumpsize* and *dumplo*, respectively, for use by **savecore**(8).

Device names specified in configuration clauses are mapped to block device major numbers with the file **devices.***machine*, where *machine* is the machine type previously specified in the configuration file. If a device name to block device major number mapping must be overridden, a device specification may be given in the form '**major** *x* **minor** *y*'.

The second group of lines in the configuration file describe which devices your system has and what they are connected to (for example, a Xylogics 450 Disk Controller at address 0xee40 in the Multibus I/O space). These lines have the following format:

dev_type dev_name **at** *con_dev more_info*

dev_type is either **controller, disk, tape, device,** or **pseudo-device.** These types have the following meanings:

controller	A disk or tape controller.
disk or **tape**	Devices connected to a controller.
device	Something "attached" to the main system bus, like a cartridge tape interface.
pseudo-device	A software subsystem or driver treated like a device driver, but without any associated hardware. Current examples are the pseudo-tty driver and various network subsystems. For pseudo-devices, **more_info** may be specified as an integer, that gives the value of the symbol defined in the header file created for that device, and is generally used to indicate the number of instances of the pseudo-device to create.

dev_name is the standard device name and unit number (if the device is not a **pseudo-device**) of the device you are specifying. For example, **xyc0** is the *dev_name* for the first Xylogics controller in a system; **ar0** names the first quarter-inch tape controller.

con_dev is what the device you are specifying is connected to. It is either nexus?, a bus type, or a controller. There are several bus types which are used by **config** and the kernel.

The different possible bus types are:

obmem	On board memory
obio	On board io
mbmem	Multibus memory (**sun2** system only)
mbio	Multibus io (**sun2** system only)
vme16d16 (vme16)	16 bit VMEbus/ 16 bit data
vme24d16 (vme24)	24 bit VMEbus/ 16 bit data
vme32d16	32 bit VMEbus/ 16 bit data (**sun3** system only)
vme16d32	16 bit VMEbus/ 32 bit data (**sun3** system only)
vme24d32	24 bit VMEbus/ 32 bit data (**sun3** system only)
vme32d32 (vme32)	32 bit VMEbus/ 32 bit data (**sun3** system only)

All of these bus types are declared to be connected to nexus. The devices are hung off these buses. If the bus is wildcarded, then the autoconfiguation code will determine if it is appropriate to probe for the device on the machine that it is running on. If the bus is numbered, then the autoconfiguation code will only look for that device on machine type N. In general, the Multibus and VMEbus bus types are always wildcarded.

more_info is a sequence of the following:

csr *address* — Specify the address of the **csr** (command and status registers) for a device. The **csr** addresses specified for the device are the addresses within the bus type specified.

The **csr** address must be specified for all controllers, and for all devices connected to a main system bus.

drive *number* — For a disk or tape, specify which drive this is.

flags *number* — These flags are made available to the device driver, and are usually read at system initialization time.

priority *level* — For devices which interrupt, specify the interrupt level at which the device operates.

vector *intr number* [*intr number* . . .]
For devices which use vectored interrupts on VMEbus systems, *intr* specify the vectored interrupt routine and *number* the corresponding vector to be used (0x40-0xFF).

A **?** may be substituted for a number in two places and the system will figure out what to fill in for the **?** when it boots. You can put question marks on a *con_dev* (for example, at virtual '?'), or on a drive number (for example, drive '?'). This allows redundancy, as a single system can be built which will boot on different hardware configurations.

The easiest way to understand **config** files it to look at a working one and modify it to suit your system. Good examples are provided in *Installing SunOS 4.1*.

FILES

Files in **/usr/share/sys/sun[2 3 4]/conf** which may be useful for developing the *config_file* used by **config** are:

GENERIC — These are generic configuration files for either a Sun-2 or Sun-3 system. They contain all possible device descriptions lines for the particular architecture.

README — File describing how to make a new kernel.

As shipped from Sun, the files used by **/usr/etc/config** as input are in the **/usr/include/sys/conf** directory:

config_file	System-specific configuration file
Makefile.src	Generic prototype makefile for Sun–[23] systems
files	List of common files required to build a basic kernel
devices	Name to major device mapping file for Sun–[23] systems

/usr/etc/config places its output files in the *../config_file* directory:

mbglue.s	Short assembly language routines used for vectored interrupts
ioconf.c	Describes I/O devices attached to the system
makefile	Used with **make**(1) to build the system
device_name.**h**	a set of header files (various *device_name*'s) containing devices which can be compiled into the system

SEE ALSO

gprof(1), **make**(1), **kgmon**(8), **savecore**(8)

The SYNOPSIS portion of each device entry in Section 4 of this manual.

Installing SunOS 4.1
System and Network Administration

NAME

crash – what happens when the system crashes

DESCRIPTION

This section explains what happens when the system crashes and how you can analyze crash dumps.

When the system crashes voluntarily, it displays a message of the form

panic: *why i gave up the ghost*

on the console, takes a dump on a mass storage peripheral, and then invokes an automatic reboot procedure as described in **reboot**(8). Unless some unexpected inconsistency is encountered in the state of the file systems due to hardware or software failure, the system will then resume multiuser operations.

The system has a large number of internal consistency checks; if one of these fails, it will panic with a very short message indicating which one failed.

When the system crashes it writes (or at least attempts to write) an image of memory into the back end of the primary swap area. After the system is rebooted, you can run the program **savecore**(8) to preserve a copy of this core image and kernel namelist for later perusal. See **savecore**(8) for details.

To analyze a dump you should begin by running **adb**(1) with the **–k** flag on the core dump, as described in *Debugging Tools*.

The most common cause of system failures is hardware failure, which can reflect itself in different ways.

See DIAGNOSTICS for some messages that you may encounter, with some hints as to causes. In each case there is a possibility that a hardware or software error produced the message in some unexpected way.

FILES

/vmunix	the system kernel
/etc/rc.local	script run when the local system starts up

SEE ALSO

adb(1), **analyze**(8), **reboot**(8) **sa**(8), **savecore**(8)

Debugging Tools

DIAGNOSTICS

IO err in push

hard IO err in swap The system encountered an error trying to write to the paging device or an error in reading critical information from a disk drive. You should fix your disk if it is broken or unreliable.

timeout table overflow

This really should not be a panic, but until the data structure is fixed, involved, running out of entries causes a crash. If this happens, you should make the timeout table bigger by changing the value of **ncallout** in the **param.c** file, and then rebuild your system.

trap type *type*, **pid** *process-id*, **pc** = *program-counter*, **sr** = *status-register*, **context** *context-number*

A unexpected trap has occurred within the system; typical trap types are:

- Bus error
- Address error
- Illegal instruction
- Divide by zero
- Chk instruction
- Trapv instruction
- Privilege violation
- Trace
- 1010 emulator trap
- 1111 emulator trap
- Stack format error

- Uninitialized interrupt
- Spurious interrupt

The favorite trap types in system crashes are "Bus error" or "Address error", indicating a wild reference. The *process-id* is the ID of the process running at the time of the fault, *program-counter* is the hexadecimal value of the program counter, *status-register* is the hexadecimal value of the status register, and *context-number* is the context that the process was running in. These problems tend to be easy to track down if they are kernel bugs since the processor stops cold, but random flakiness seems to cause this sometimes.

init died The system initialization process has exited. This is bad news, as no new users will then be able to log in. Rebooting is the only fix, so the system just does it right away.

NAME

cron – clock daemon

SYNOPSIS

/usr/etc/cron

DESCRIPTION

cron executes commands at specified dates and times. Regularly scheduled commands can be specified according to instructions found in **crontab** files in the directory **/var/spool/cron/crontabs**. Users can submit their own **crontab** files using the **crontab**(1) command. Commands that are to be executed only once may be submitted using the **at**(1) command.

cron only examines **crontab** files and **at** command files during process initialization and when a file changes using **crontab** or **at**. This reduces the overhead of checking for new or changed files at regularly scheduled intervals.

Since **cron** never exits, it should only be executed once. This is normally done by running **cron** from the initialization process through the file **/etc/rc**; see **init**(8). **/var/spool/cron/FIFO** is a FIFO file that **crontab** and **at** use to communicate with **cron**; it is also used as a lock file to prevent the execution of more than one **cron**.

FILES

/var/spool/cron	main cron directory
/var/spool/cron/FIFO	FIFO for sending messages to **cron**
/var/spool/cron/crontabs	directory containing **crontab** files

SEE ALSO

at(1), **crontab**(1), **sh**(1), **queuedefs**(5), **init**(8), **syslogd**(8)

DIAGNOSTICS

cron logs various errors to the system log daemon, **syslogd**(8), with a facility code of **cron**. The messages are listed here, grouped by severity level.

Err Severity

Can't create /var/spool/cron/FIFO: *reason*
cron was unable to start up because it could not create **/var/spool/cron/FIFO**.

Can't access /var/spool/cron/FIFO: *reason*
cron was unable to start up because it could not access **/var/spool/cron/FIFO**.

Can't open /var/spool/cron/FIFO: *reason*
cron was unable to start up because it could not open **/var/spool/cron/FIFO**.

Can't start cron - another cron may be running (/var/spool/cron/FIFO exists)
cron found that **/var/spool/cron/FIFO** already existed when it was started; this normally means that **cron** had already been started, but it may mean that an earlier **cron** terminated abnormally without removing **/var/spool/cron/FIFO**.

Can't stat /var/spool/cron/FIFO: *reason*
cron could not get the status of **/var/spool/cron/FIFO**.

Can't change directory to *directory*:*reason*
cron could not change to *directory*.

Can't read *directory*:*reason*
cron could not read *directory*.

error reading message: *reason*
An error occurred when **cron** tried to read a control message from **/var/spool/cron/FIFO**.

message received — bad format
A message was successfully read by **cron** from **/var/spool/cron/FIFO**, but the message was not of a form recognized by **cron**.

SIGTERM
received **cron** was told to terminate by having a SIGTERM signal sent to it.

cron could not unlink /var/spool/cron/FIFO: *reason*
cron was told to terminate, but it was unable to unlink **/var/spool/cron/FIFO** before it terminated.

********* CRON ABORTED **********
cron terminated, either due to an error or because it was told to.

Can't open queuedefs file *file*:*reason*
cron could not open a *queuedefs* file.

I/O error reading queuedefs file *file*:*reason*
An I/O error occurred while **cron** was reading a *queuedefs* file.

Using default queue definitions
An error occurred while trying to read a *queuedefs* file; the default queue definitions will be used.

Can't allocate *number* **bytes of space**
An internal error occurred in **cron** while trying to allocate memory.

Info Severity

queue **queue max run limit reached**
There were more jobs running or to be run in the queue *queue* than the maximum number specified. **cron** will wait until one of the currently-running jobs completes before starting to run a new one.

MAXRUN (25) procs reached
There were more than 25 jobs running or to be run by **cron**. **cron** will wait until one of the currently-running jobs completes before starting to run a new one.

***** cron started *****
cron started running.

> CMD: *command*
A **cron** job was started. *command* is the command to be run.

> *user pid queue time*
A **cron** job was started for user *user*, in queue *queue*, with process ID *pid*, at the date and time *time*.

< *user pid queue time status*
A **cron** job completed for user *user*, in queue *queue*, with process ID *pid*, at the date and time *time*. If the command terminated with a non-zero exit status or a signal, *status* indicates the exit status or signal.

Notice Severity

Can't fork
An attempt to **fork** (2) to run a new job failed; **cron** will attempt again after a 30-second delay.

Warning Severity

Can't stat queuedefs file *file*:*reason*
cron could not get the status of a *queuedefs* file in order to determine whether it has changed. **cron** will assume it has changed and will reread it.

NAME

dcheck – file system directory consistency check

SYNOPSIS

/usr/etc/dcheck [**–i** *numbers*] [*filesystem*]

DESCRIPTION

Note: **dcheck** has been superceded for normal consistency checking by **fsck**(8).

dcheck reads the directories in a file system and compares the link-count in each inode with the number of directory entries by which it is referenced. If the file system is not specified, **dcheck** checks a set of default file systems.

dcheck is fastest if the raw version of the special file is used, since the i-list is read in large chunks.

OPTIONS

–i *numbers*

numbers is a list of i-numbers; when one of those i-numbers turns up in a directory, the number, the i-number of the directory, and the name of the entry are reported.

FILES

Default file systems vary with installation.

SEE ALSO

fs(5), **fsck**(8), **clri**(8), **icheck**(8), **ncheck**(8)

DIAGNOSTICS

When a file turns up for which the link-count and the number of directory entries disagree, the relevant facts are reported. Allocated files which have 0 link-count and no entries are also listed. The only dangerous situation occurs when there are more entries than links; if entries are removed, so the link-count drops to 0, the remaining entries point to thin air. They should be removed. When there are more links than entries, or there is an allocated file with neither links nor entries, some disk space may be lost but the situation will not degenerate.

BUGS

Since **dcheck** is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

Inode numbers less than 2 are invalid.

NAME

eeprom – EEPROM display and load utility

SYNOPSIS

eeprom [**–**] [**–i**] [**–f** *filename*] [*field* [=*value*]] ...

eeprom [**–i**] [**–c**] [**–f** *filename*]

AVAILABILITY

Not available for Sun-2 systems.

DESCRIPTION

eeprom displays or changes the values of fields in the EEPROM. It processes fields in the order given. When processing a *field* accompanied by a *value*, **eeprom** makes the indicated alteration to the EEPROM; otherwise it displays the *field*'s value. When given no field specifiers, **eeprom** displays the values of all EEPROM fields. The '**—**' option specifies that fields and values are to be read from stdin (one *field* or *field*=*value* per line).

eeprom verifies the EEPROM checksums and complains if they are incorrect; if the **–i** flag is specified, erroneous checksums are ignored. If the **–c** flag is specified, all incorrect checksums are recomputed and corrected in the EEPROM.

OPTIONS

–i	Ignore bad checksums.
–f *filename*	Use *filename* as the EEPROM device.
–c	Correct bad checksums.
—	Read field names and values from stdin.

The field names and their possible values are:

hwupdate	a valid date (including "today" and "now")
memsize	8 bit decimal integer (megabytes of memory on machine)
memtest	8 bit decimal integer (megabytes of memory to test)
scrsize	"1024x1024", "1152x900", "1600x1280", or "1440x1440"
watchdog_reboot	"true" or "false"
default_boot	"true" or "false"
bootdev	*charchar*(*hex-int*,*hex-int*,*hex-int*) (with *char* a character, and *hex-int* a hexadecimal integer.)
kbdtype	8 bit decimal integer (0 for all Sun keyboards)
keyclick	"true" or "false"
console	"b&w" or "ttya" or "ttyb" or "color"
custom_logo	"true" or "false"
banner	banner string
diagdev	%c%c (%x,%x,%x) - diagnostic boot device
diagpath	diagnostic boot path
ttya_no_rtsdtr	"true" or "false"
ttyb_no_rtsdtr	"true" or "false"
ttya_use_baud	"true" or "false"
ttyb_use_baud	"true" or "false"
ttya_baud	baud rate (16-bit decimal integer)
ttyb_baud	baud rate (16-bit decimal integer)
columns	number of columns on screen (8-bit decimal integer)
rows	number of rows on screen (8-bit decimal integer)

FILES

/dev/eeprom

SEE ALSO

mon/eeprom.h

NAME

etherd, rpc.etherd – Ethernet statistics server

SYNOPSIS

/usr/etc/rpc.etherd *interface*

AVAILABILITY

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

etherd is a server which puts *interface* into promiscuous mode, and keeps summary statistics of all the packets received on that interface. It responds to RPC requests for the summary. You must be root to run **etherd**.

interface is a networking interface such as **ie0**, **ie1**, **ec0**, **ec1** and **le0**.

traffic(1C) displays the information obtained from **etherd** in graphical form.

SEE ALSO

traffic(1C)

NAME

iostat – report I/O statistics

SYNOPSIS

iostat [*interval* [*count*]]

DESCRIPTION

iostat iteratively reports the number of characters read and written to terminals, and, for each disk, the number of kilobytes transferred per second, and the milliseconds per average seek. It also gives the percentage of time the system has spent in user mode, in user mode running low priority (niced) processes, in system mode, and idling.

To compute this information, for each disk, seeks and data transfer completions and number of words transferred are counted; for terminals collectively, the number of input and output characters are counted. Also, each fiftieth of a second, the state of each disk is examined and a tally is made if the disk is active. From these numbers and given the transfer rates of the devices approximate average seek times are calculated for each device.

If *interval* is specified, **iostat** reports once each *interval* seconds. The first report is for all time since a reboot and each subsequent report is for the last interval only.

count restricts the number of reports.

FILES

/dev/kmem
/vmunix

SEE ALSO

vmstat(8)

BUGS

iostat only provides statistics on the first four disk drives, all others are ignored.

NAME

ipallocd – Ethernet-to-IP address allocator

SYNOPSIS

/usr/etc/rpc.ipallocd

AVAILABILITY

Sun386i systems only.

DESCRIPTION

ipallocd is a daemon that determines or temporarily allocates IP addresses within a network segment. It has complete knowledge of the hosts listed in the yellow pages, and, if the system is running the name server, of any hosts listed in internet domain tables automatically accessed on that host through the standard library *gethostbyaddr( )* call.

This protocol uses DES authentication (the Sun Secure RPC protocol) to restrict access to this function. The only clients privileged to allocate addresses are those whose net IDs are in the *networks* group. For machine IDs, the machine must be a YP server.

The daemon uses permanent entries in the */etc/ethers* and */etc/hosts* files when they exist and are usable. In other cases, such as when a system is new to the network, *ipallocd* will enter a temporary mapping in a local cache. Entries in the cache are removed when there have been no references to a given entry in a 24-hour period. This cache survives system crashes so that IP addresses will remain consistent.

The daemon also provides corresponding IP address to name mapping.

ipallocd refuses to allocate addresses on networks not listed in the *netrange* file, or for which no free address is available.

FILES

/etc/ez/ipalloc.cache
/etc/ez/ipalloc.netrange

SEE ALSO

pnp(3R), **ipalloc**(3R), **ipallocd**(8C), **pnpboot**(8C), **netconfig**(8C)

NAME

vmstat – report virtual memory statistics

SYNOPSIS

vmstat [**–cfisS**] [*interval* [*count*]]

DESCRIPTION

vmstat delves into the system and normally reports certain statistics kept about process, virtual memory, disk, trap and CPU activity.

Without options, **vmstat** displays a one-line summary of the virtual memory activity since the system has been booted. If *interval* is specified, **vmstat** summarizes activity over the last *interval* seconds. If a *count* is given, the statistics are repeated *count* times.

For example, the following command displays a summary of what the system is doing every five seconds. This is a good choice of printing interval since this is how often some of the statistics are sampled in the system.

```
example% vmstat 5
 procs   memory            page    faults
 r b w  avm  fre  re at  pi po fr  de sr x0 x1 x2 x3  in  sy  cs us sy id
 2 0 0  918  286  0  0   0  0  0   0  1  0  0  0  0   4  12   5  3  5 91
 1 0 0  846  254  0  0   0  0  0   0  6  0  1  0  0  42 153  31  7 40 54
 1 0 0  840  268  0  0   0  0  0   0  5  0  0  0  0  27 103  25  8 26 66
 1 0 0  620  312  0  0   0  0  0   0  6  0  0  0  0  26  76  25  6 27 67
^C
example%
```

The fields of **vmstat**'s display are:

procs Report the number of processes in each of the three following states:

- **r** in run queue
- **b** blocked for resources (i/o, paging, etc.)
- **w** runnable or short sleeper (< 20 secs) but swapped

memory Report on usage of virtual and real memory. Virtual memory is considered active if it belongs to processes which are running or have run in the last 20 seconds.

- **avm** number of active virtual Kbytes
- **fre** size of the free list in Kbytes

page Report information about page faults and paging activity. The information on each of the following activities is averaged each five seconds, and given in units per second.

- **re** page reclaims — but see the –S option for how this field is modified.
- **at** number of attaches — but see the –S option for how this field is modified.
- **pi** kilobytes per second paged in
- **po** kilobytes per second paged out
- **fr** kilobytes freed per second
- **de** anticipated short term memory shortfall in Kbytes
- **sr** pages scanned by clock algorithm, per-second

disk Report number of disk operations per second (this field is system dependent). For Sun systems, four slots are available for up to four drives: "x0" (or "s0" for SCSI disks), "x1", "x2", and "x3".

faults Report trap/interrupt rate averages per second over last 5 seconds.

- **in** (non clock) device interrupts per second
- **sy** system calls per second
- **cs** CPU context switch rate (switches/sec)

cpu Give a breakdown of percentage usage of CPU time.
- **us** user time for normal and low priority processes
- **sy** system time
- **id** CPU idle

OPTIONS

–c Report cache flushing statistics. By default, report the total number of each kind of cache flushed since boot time. The types are: user, context, region, segment, page, and partial-page.

–f Report on the number of **forks** and **vforks** since system startup and the number of pages of virtual memory involved in each kind of fork.

–i Report the number of interrupts per device. Autovectored interrupts (including the clock) are listed first.

–s Display the contents of the **sum** structure, giving the total number of several kinds of paging-related events which have occurred since boot.

–S Report on swapping rather than paging activity. This option will change two fields in **vmstat**'s ''paging'' display: rather than the ''re'' and ''at'' fields, **vmstat** will report ''si'' (swap-ins), and ''so'' (swap-outs).

FILES

/dev/kmem
/vmunix

BUGS

If more than one autovectored device has the same name, interrupts are counted for all like-named devices regardless of unit number. Such devices are listed with a unit number of '?'.

Index

1

1/4" tape drive, 69
150-Mbyte 1/4" tape drive, 69
150-Mbyte tape, format addressing, 71

2

2.3-Gbyte 8mm tape drive, 69
2.3-Gbyte, format addressing, 71

4

4.0.3 installation tape, 17, 20
4.0.3 tape contents, 38

8

8mm tape drive, 69

A

adding other architectures after installation, 29
address formats, tape, 71
addressing, IPI, 57
ALM-2, adjusting kernel, 50
architecture, primary, 20
architecture, Sun-4, 20
architectures, adding after installation, 29
architectures, other, 22

B

boot address format, IPI, 58
boot address formats, tape, 72
boot program, 13
boot vmunix, 16
boot, network, 35
booting from IPI disk, 58
booting in single-user mode, 26
booting multi-user, 28
booting the miniroot, 15
booting, more than one tape mounted, 16
booting, setting EEPROM for, 26
bootparams, 33
building a kernel, 21

C

changing size of root partition, 19
changing size of swap partition, 19
channel, definition, 57
client architectures, adding after installation, 29
client form, 22
client usr partitions, 24
client, creating temporary, 31
configuring a kernel, 45
configuring an Ethernet gateway, 27, 75
contents, release documentation box, 6
contents, tape, 38
custom kernel, 27
customized kernel, need for Sys category, 21
customizing a kernel, 45, 50

D

data format addressing, tape, 71
dataless client installation, 28
dd command, 34
default boot from IPI disk, 58
defect lists, 15
definition of terms, IPI, 57
device names, IPI, 58
device names, table, 70
device names, tape drive, 69
device type, mt command for determining, 70
dialbox, need for SunView software, 21
disk defect lists, 15
disk error messages, 61, 62, 63
disk form, 19, 20
disk form, preserve option, 19
disk power-cycle, need to reboot, 61
disk space, planning, 12
disk swap, need to reboot, 61
disk, booting from IPI, 58
disks, assigning some, 20
disks, formatting, partitioning, labeling, 12
documentation box contents, 6

E

eeprom command, modifying boot disk, 59
EEPROM q command, 58
eeprom, check current settings, 59
eeprom, modifying, 59
EEPROM, modifying for boot, 58
EEPROM, setting for IPI booting, 26

encryption tape, Sun-4c, 29
error message, tape device, 21
error messages, IPI, 61, 62, 63
error messages, kernel configuration, 52
etc/bootparams, 33
etc/ttytab, see ttytab, 51
Ethernet address, 31
Ethernet controller, second, 75
Ethernet gateway, 27
Ethernet gateway, configuring, 75
Ethernet, second, 18
exec directories, 24
export/swap, 19
extract list, 21
extracting tape TOC, 40

F

facility, definition, 57
first-time installation, 19
FLT, format addressing, 71
format menu, 15
format program, 14
formatting disks, 12, 14
freehog partition, 19, 20, 21

G

gateway, Ethernet, 27
GENERIC kernel, 27, 45

H

halting miniroot, 26
heterogeneous installation, 19
host form, 18

I

IDST490 kernel, 27, 45
increasing freehog partition, 19
install small kernel, 26
installation complete, 23
installation overview, 11
installation, dataless client, 28
installation, first-time, 19
installation, heterogeneous, 19
installation, local, 12
installation, remote, 31
installation, startup, 23
installing SPARCserver 490 software, 11
internet address, 31
IPI addressing, 57
IPI boot address format, 58
IPI configuration options, 57
IPI controller and disk drives, 57
IPI device names, 58
IPI disk, booting from, 58
IPI error messages, 61, 62, 63
IPI naming conventions, 57
IPI physical addresses, 57
IPI, definition of terms, 57
IPI8-1000 disk drives, 57
ISP-80 controller, 57

K

kernel configuration, 45
kernel configuration files, 45
kernel configuration, error messages, 52
kernel configuration, monitoring system performance, 52
kernel configuration, number of ports, 50
kernel, adjusting maxusers, 50
kernel, custom, 50
kernel, customized, 21
kernel, customizing, 27
kernel, GENERIC, 27, 45
kernel, IDST490, 27, 45
kernel, install small, 26
kernel, making ttytab entries, 51

L

labeling disks, 12
local installation, 12

M

machine architecture terminology, 8
man pages, 95
maxusers, adjusting in kernel, 50
mcpa, adjusting in kernel, 50
media box contents, 4
miniroot, 15, 16
miniroot, adding client architectures, 30
miniroot, copy to remotehost, 33
miniroot, do not copy to partition "a", 16
miniroot, halt, 26
miniroot, warning about copying, 33
modifying boot disk, eeprom command, 59
modifying EEPROM for booting, 58
monitoring system performance, 52
mt command, 34
mt command, determining device type, 70
multi-user, booting, 28
munix, 12, 13
munix file system, 13

N

naming conventions, IPI, 57
netmask, 27
network boot, 35
network port, checking state, 75
newfs, remote install, 36
next steps, 25

P

partition "a", 16
partitioning disks, 12
physical addresses, IPI, 57
planning disk space, 12
ports, adjusting number of, 50
post-installation procedures, 25, 26

power-cycle disk, need to reboot, 61
preserve option, 19
previous version of vmunix, 31
primary architecture, 20

Q

q command, EEPROM, 58

R

rebooting after disk swap, 61
reducing /export/swap partition, 19
reinstalling Sun-4c, 30
release documentation box contents, 6
remote install, format disks, 36
remote install, mounting miniroot, 34
remote install, remove client, 37
remote install, restore file system, 36
remote install, restore spare disk partition, 36
remote install, run SunInstall, 36
remote install, run sunupgrade, 36
remote installation, 31
remote installation, spare disk required, 31
remotehost, 31
remove client, 37
requirement for remote installation, 31
restore spare disk partition, 36
reverting to previous vmunix, 31
root partition, 19

S

saving vmunix, 31
SCSI configuration options, 69
SCSI devices, 69
SCSI tape drives, 69
second Ethernet, 18
second Ethernet controller, 75
second Ethernet port, 27
serial ports, number in kernel, 50
server, need for client form, 22
set EEPROM for IPI booting, 26
setup_client, 30, 31
setup_client, completed, 32
setup_client, remove client, 37
setup_client, running, 32
setup_exec, 29
single-user mode, 26
slave, definition, 57
small kernel, install, 26
software categories, 20, 21
software categories, listing files, 41
software extract list, 21
software form, 20
software form, other architectures, 22
software form, setup_exec, 29
software support, adding after installation, 29
software, installation, 11
SPARCserver 490 description, 3
SPARCserver 490 tape, 23
spare disk for remote installation, 31
standalone copy, 16
starting SunInstall, 17
starting the installation, 23
subnetting, 27
Sun-2/50, small kernel, 26
Sun-3/50, small kernel, 26
Sun-3/60, small kernel, 26
Sun-3/80, small kernel, 26
Sun-4 architecture, 20
Sun-4/110, small kernel, 26
Sun-4/330, small kernel, 26
Sun-4c encryption tape, 29
Sun-4c, add last, 29
Sun-4c, install last, 22
Sun-4c, need to reinstall, 30
SunDials, need for SunView software, 21
SunInstall, 17
SunInstall disk form, 19
SunInstall host form, 18
SunInstall Main Menu, 17
SunNet Ethernet controller, 27
SunNet Ethernet/VME Controller, 75
sunupgrade, 23
sunupgrade assumptions, 24
sunupgrade complete, 25
sunupgrade example, 23, 24
sunupgrade, after adding architectures, 30
sunupgrade, next steps, 25
SunView software, 21
swap partition, 19
Sys category, need for custom kernel, 21

T

tape contents, 38
tape contents, 4.0.3, 38
tape contents, extracting TOC, 40
tape contents, listing files in software category, 41
tape contents, SPARCserver 490, 39
tape drive device names, 69
tape drive, data format addressing, 71
tape drives, 69
tape, boot address formats, 72
tape, SPARCserver 490, 23
terminology, machine architecture, 8
tftpboot, 33, 35
TOC, tape, 40
ttytab, making entries, 51
ttytab, sample entries, 51
typographic conventions, 7

U

usr partition, 19, 21

V

vmunix, booting, 16
vmunix, previous version, 31

W

warning about copying miniroot, 33

Y

yp client, 27, 28
yp maps, 28
yp name service, 12, 26, 28, 31, 32
yp server, 27
ypbind, 28